Ricardo Vaz de Carvalho Neto

Licenciado em Ciências de Engenharia Eletrotécnica e de Computadores

Implementação de um Filtro FIR numa FPGA para processamento de imagem

Dissertação para obtenção do Grau de Mestre em

Engenharia Electrotécnica e de Computadores

Orientador: Prof. Dr. Luís Augusto Bica Gomes de Oliveira, Prof. Auxiliar com Agregação, Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Setembro, 2019

Implementação de um Filtro FIR numa FPGA para processamento de imagem

Este documento foi gerado utilizando o processador (pdf)LaTeX, com base no template "novathesis" [1] desenvolvido no Dep. Informática da FCT-NOVA [2].
[1] https://github.com/joaomlourenco/novathesis [2] http://www.di.fct.unl.pt

Emília.

Agradecimentos

A maior dificuldade no meu percurso fui eu próprio, desde o primeiro instante. Desde que me lembro, o meu medo de falhar, desiludir, magoar ou ser magoado, tem permeado cada instância da minha vida e feito com que me isole e não tente. Ao fazê-lo, estagnei e não só afastei, com receio, o "mau" da minha vida - como também o bom. Reflectindo agora nestes anos passados, terminar o meu percurso académico deu-me uma sensação de orgulho e feito realizado que durante demasiado tempo não achei que fosse possível, portanto uso agora este espaço para agradecer a todas as pequenas grandes peças que ajudaram a montar este puzzle.

Lembramos a vida pelos momentos marcantes, mas o sublime está no nosso dia-a-dia e nos entre-tantos. Nas pequenas escolhas que se faz quando ninguém está a ver. E no trabalho que se desenvolve em prol do nosso crescimento pessoal.

Assim, deixo o meu agradecimento à Faculdade de Ciências e Tecnologias da Universidade Nova de Lisboa, pois estou convicto que não poderia ter escolhido uma melhor instituição para a minha formação, e a todos elementos do corpo docente do Departamento de Engenharia Electrotécnica com quem me cruzei, por todos os ensinamentos que me foram transmitidos a nível pessoal e académico. Ao meu orientador, professor Luís Oliveira, agradeço pela disponibilidade, paciência e acima de tudo versatilidade ao ajudar-me a arranjar um tema para poder desenvolver com gosto e pelo apoio prestado ao longo deste trabalho. O seu constante entusiasmo perante os temas tratados e facilidade em ajudar tornaram todo o processo mais dinâmico e serão sempre recordados. Ao Diogo Boto que me deu uma ajuda valiosa nesta fase final do trabalho.

Aos incansáveis *Mad Lads*, Daniel "*Dan-O*" Rodrigues, Filipe "*Senpai*" Gonçalves, Luís "*Loki*" Severo e Octávio "*Xerife*" Rosa, pela motivação constante e apoio nesta etapa. Do estoicismo a *go-getters*, "Even though I walk through the valley of the shadow of death, I will fear no failure, for the lads are with me". *Tamu junto*.

À Ana Carreira, André Gomes, Catarina Martins, Fábio Pereira, Fábio Vidago, Gonçalo Cabrita, João Carvalho, João Marques, João de Oliveira, José Ferreira, Rúben Carvalho , Rui Calado e Stephany Tomé, *brothers in arms*.

Ao Andy Gonçalves, Daniela Freire, Diogo Gouveia, Eliana Carvalho, Flávio Moreira, Gonçalo Homem, Inês Mendes, Inês Rico, Kaddi Dores, Margarida Andrade, Nuno Correia, Pedro Verdelho, Ricardo Deus, Sandro Nunes, Sasha Fonseca, Sara Andrade, Sofia Sousa, Tiago Chá e Teresa Coelho. *Bezanas*.

Ao Júlio Bartolomeu, Pedro Casanova, Pedro Zoio e Stylianos Neocleous, os que guardo de Sampaio.

À Jannett Neves, Mário Ilhéu e Ricardo Mestre por me esgotarem o juízo.

À Andreia Ribeiro e ao Rodrigo Francisco, pela fonte inesgotável de amor e paciência.

À Drª Júlia Murta e à Teresa Brissos, que carregaram a minha consciência às costas. *AlErui*.

Aos meus avós, Maria Amélia, Emília e José Neto.

Aos meus sobrinhos Laura, Morris, Ernest, e primos Igor, Sofia e Sérgio. Por fim à minha irmã Catarina Brandão e aos meus pais, José e Maria Neto. É por eles que vale a pena.

Resumo

Há demasiado tempo que o flagelo dos incêndios em Portugal anualmente assola o nosso país e noticiários, levando consigo acima de tudo vidas, terras e todo o tipo de recursos naturais, enquanto dizima o património natural e contribui directamente para um aumento do efeito de estufa.

Esta dissertação representa uma forma de atacar o problema numa vertente de recursos tecnológicos e longe do terreno que poderá um dia vir a ser implementadas numa fase tardia, solução essa que passou pelo uso das FPGA, e explorar o seu possível uso numa vertente de processamento de imagens. Neste caso com imagens de incêndios, com o intuito de fazer uma análise inicial de existir - ou não - fogo na imagem.

Assim apresenta-se um estudo da tecnologia FPGA e de seguida, implementação nesta de uma estrutura de um filtro FIR de ordem e parâmetros manipuláveis, que será um dos blocos de construção para a eventual instalação futura de um algoritmo de processamento de imagem.

É também apresentada uma metodologia, pertinente à placa FPGA escolhida, com o objectivo de ajudar futuros utilizadores do material de forma a não estarem sujeitos a uma curva de aprendizagem igualmente íngreme, sendo apresentados todos os passos efectuados no desenvolvimento do protótipo e respectivos resultados da implementação.

Palavras-chave: Arduino; FPGA; filtro digital; filtro FIR; passa baixo; processamento de imagem

Abstract

The scourge of forest fires in Portugal has been plaguing the country and our headlines every year for far too long, taking with it lives, lands and all kinds of natural resources, while decimating our natural heritage and directly contributing to an increase in the greenhouse effect.

This dissertation represents a way of tackling the problem through the use of technological resources and within a safe distance that may one day be implemented at a later stage, which was the use of FPGA, and to explore its possible use in an image processing situation. In this case forest fires, in order to make an initial analysis of whether or not there is a fire in the image.

Thus is presented a study of the FPGA technology, followed by the implementation of a FIR filter structure of selectable order and parameters, which will be one of the building blocks for the eventual future installation of an image processing algorithm.

A methodology, relevant to the chosen FPGA board, is also presented, with the aim of supporting future users of the material so that they are not subject to an equally steep learning curve, presenting all the steps taken in the prototype development and their implementation results.

Keywords: Arduino; FPGA; digital filter; FIR filter; low pass; image processing

Índice

Lista de Figuras

Lista de Tabelas

Listagens

Capítulo 1

Introdução

Neste capítulo, é realizada uma breve introdução sobre os temas relacionados com o trabalho a ser desenvolvido.

Para tal, procede-se à identificação do problema principal a ser atacado, da motivação para a sua concretização e a contribuição que resultará desta sua implementação. No fim, é apresentada a estrutura da presente dissertação.

1.1 Identificação do Problema e Motivação

Em todo o mundo, os incêndios florestais estão fora de controle - não por causa das conflagrações que regularmente assolam os nossos noticiários, mas essencialmente porque governos, agências internacionais e em particular, as comunidades, ainda falham em chegar a um consenso sobre a forma como os incêndios em territórios nacionais devem ser combatidos e como gerir os recursos para o fazer. Isto porque o público geral e as pessoas em posição de poder são melhores a reagir a crises recorrentes de curto prazo do que a concentrar os recursos em soluções sustentáveis e de longo prazo.

Essa falta de clareza, que é tornada mais intransigente devido aos riscos de incêndio ou inerentes ao fogo serem usados para promover uma complexa mistura de interesses privados e contraditórios (que organizações combatem o quê, que equipamento a usar, as compensações monetárias ao deixar um fogo alastrar, etc), o que significa que continuarão a ser uma fonte de controvérsia, gastos e danos no futuro da humanidade.

Em Portugal eles são a catástrofe natural recorrente mais grave, só nos anos de 2003 e 2005, arderam cerca de 425.839 e 339.089 hectares respectivamente em território nacional [1], com anos como o de 2014 a apresentar uma área ardida menor de 19.930 hectares a serem a rara excepção.

Anos	Nº de incêndios rurais	Área ardida (ha)			
		Povoamentos	Matos	Agrícola	Total
2009	12.666	6.836	16.545	1.047	24.428
2010	10.354	11.774	13.706	1.860	27.340
2011	12.940	7.490	13.158	1.612	22.260
2012	15.739	25.504	41.637	5.196	72.337
2013	8.505	7.045	17.059	3.591	27.695
2014	5.537	3.285	4.218	1.409	8.912
2015	12.024	14.115	14.450	2.640	31.205
2016	5.593	3.890	4.149	1.643	9.682
2017	10.139	79.440	56.022	11.046	146.508
2018	6.275	2.072	3.101	587	5.760
2019	**6.498**	**12.908**	**8.037**	**2.968**	**23.913**
Média 2009-2018	9.977	16.145	18.405	3.063	37.613

Tabela 1.1: Número de incêndios rurais e extensão da área ardida em Portugal continental, entre 1 de Janeiro a 5 de Agosto. Dados relativos a 2019 são provisórios [2]

A partir de 2016, os incêndios voltam a Portugal com grande intensidade, quando até meados de Agosto já tinham ardido mais de 100,000 hectares, tratando-se de uma área afectada superior a metade de toda a área queimada devido a fogos na Europa nesse mesmo ano [1].

Este trabalho surge assim do intuito de contribuir com uma pequena solução tecnológica, dentro de um projecto maior, para a prevenção de incêndios em zonas de território nacional cuja cobertura, tanto de rede como recursos humanos, é fraca. Estas zonas são de difícil acesso e onde muitas vezes é impossível manter vigilância constante, visto que a visibilidade não é perfeitamente horizontal e torna-se difícil ter noção da exactidão do foco de incêndio.

1.2 Contribuição da Tese

Esta dissertação visa a ser um passo inicial para o desenvolvimento de um algoritmo de processamento de imagem digital a implementar numa FPGA, ou *field programmable gate array*, Arduino MKR Vidor 4000, com recurso a um filtro FIR.

Tal permite aplicação de algoritmos à imagem a ser processada de para retirar o máximo proveito da informação disponível, ignorando problemas como o ruído nela embutido e a eventual distorção de sinal que poderia suceder com processamento de imagens analógicas. No entanto, este futuro algoritmo visa a ser um processo simples no terreno e não uma classificação extensa.

A FPGA surge assim como solução tecnológica ao reconhecimento e controlo do terreno actuando longe do local do fogo, com uma avaliação precisa e rápida que dispensaria

a presença de homens no terreno, especialmente em concelhos do país com vários hectares e floresta não ordenada em que se torna difícil a alocação de meios humanos no terreno.

No entanto, o foco desta dissertação é apenas do manuseamento da tecnologia, as suas restrições e respectiva implementação, sendo que a elaboração do algoritmo não é abordada.

1.3 Estrutura do Documento

Este documento encontra-se dividido em 5 capítulos: Introdução, Fundamentos Tecnológicos, Estado da Arte, Implementação de um Filtro Configurável, Conclusões e Trabalho Futuro.

Em Fundamentos Tecnológicos, segue-se um estudo sobre esta tecnologia, as suas características, vantagens e desvantagens face aos competidores e dificuldades inerentes à sua arquitectura. Por fim tem-se uma análise sobre a placa seleccionada, a sua estrutura e funcionamento, bem como de um resumo sobre os filtros a lidar.

No Estado da Arte, é feito um resumo de algum do trabalho já existente na literatura e das implementações de filtros feitas em variados tipos de FPGA.

Implementação de um Filtro Configurável, demonstra-se os passos para a execução do protótipo de um filtro, desde a familiarização com o material bem como o seu desenvolvimento e todos os passos tomados para o enquadramento do problema.

Nas Conclusões e Trabalho Futuro, é feito um resumo das ilações tiradas a partir dos resultados e o que poderia ser melhorado, com uma apresentação de conclusões sobre a qualidade e viabilidade do trabalho desenvolvido, abordando-se ainda uma possível continuidade de trabalhos.

Capítulo 2

Fundamentos Tecnológicos

Este capítulo irá focar-se no estudo da tecnologias FPGA, o seu desenvolvimento face ao panorama do mercado dos processadores bem como sua acessibilidade e *learning curve* inerente ao seu uso. Para isso é feita uma análise por camadas a cada componente que constitui a arquitectura desta tecnologia, uma análise qualitativa das suas faculdades e por fim uma comparação com outras tecnologias.

Por último, será feita uma análise separada às *hardware description languages* e um foco sobre a placa seleccionada para desenvolver o trabalho desta dissertação, seguida de uma breve análise sobre os filtros digitais a ser implementados e o seu funcionamento.

2.1 *Field Programmable Gate Array*

A quantidade de dados que a humanidade produz e tem de gerir está a explodir. Este número tem vindo a duplicar a quase cada dois anos, e como é possível ver na figura 2.1, está previsto que este crescimento exponencial se mantenha em anos vindouros.

Como consequência deste crescimento, há uma necessidade cada vez maior de retirar o máximo de informação possível destes dados com baixa latência e de os usar em tempo real, o que levou a um aumento drástico das necessidades da capacidade de processamento na nossa tecnologia.

Com os CPUs a lutar só para tentar acompanhar as exigências da Lei de Moore como visto na figura 2.2 e a serem *overclocked* até aos seus limites, tem sido um desafio cada vez maior seguir o crescimento explosivo de dados produzidos, levando a uma crescente necessidade de alternativas tecnológicas capazes de lidar com o problema. Face a esta lacuna, a FPGA surge assim como uma alternativa flexível, reprogramável no terreno e eficaz.

A tecnologia nasce nos anos 80 quando os dois maiores produtores actuais de FPGA (ALtera e Xilinx) foram fundados, tendo sido usada desde há quase 40 anos para acelerar o desenvolvimentos de soluções tecnológicas baseadas em *chips*, oferecendo um ciclo de desenvolvimento muito menos demorado que os ASIC, com um risco de investimento muito menor devido à capacidade de ser reprogramada [3].

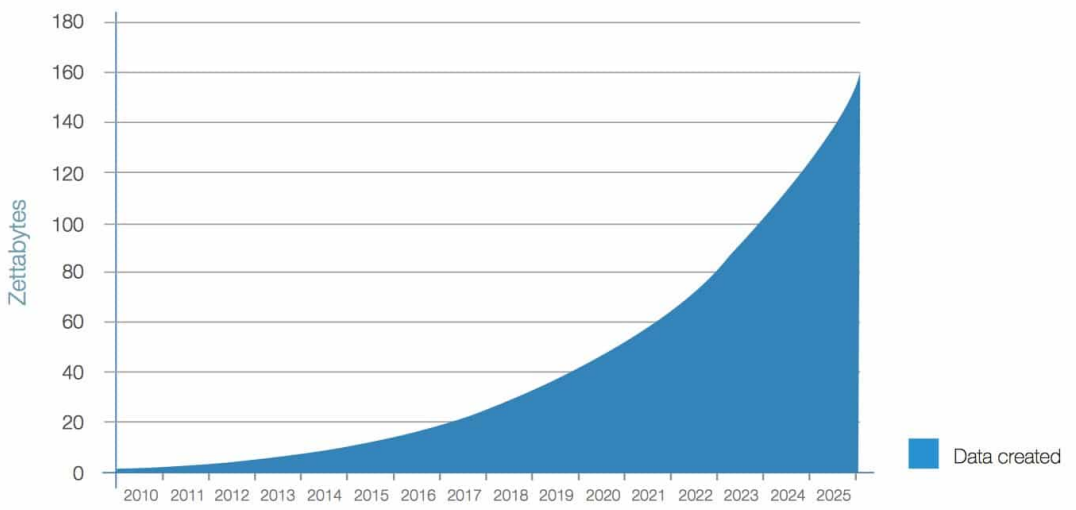

Figura 2.1: O conjunto global de dados ao longo dos anos [4]

Uma FPGA, ou *Field Programmable Gate Array*, é na sua essência um dispositivo programável e a maior aproximação possível de um produto comercial que possibilite desenhar de raiz o seu próprio chip. Da mesma forma que um ASIC – *application specific integrated circuit* - é um *micro chip* desenhado para cumprir uma função ou aplicação em particular, tais como um protocolo de transmissão específico. As FPGA são assim o oposto de circuitos integrados comuns, como os micro processadores ou as memórias RAM em cada computador pessoal [3].

Esta tecnologia permite desenhar e implementar virtualmente qualquer função digital que possa ser possível imaginar e conceber dentro de um único chip universal, sendo a principal diferença entre uma FPGA e qualquer outro chip semelhante é que esta não faz efectivamente nada por si só - não tem nenhum propósito nem função incutida nele no momento em que chega às mãos dos utilizadores.

Esta característica particular contrasta ainda mais as FPGA com um micro-controlador, que é na verdade um pequeno dispositivo de um computador, contendo já em si toda a lógica *hardwired* e ligações necessárias para poder funcionar, precisando apenas do *input* de um programa.

Com uma FPGA tal não é possível, visto que ao ser inicializada com os seus *factory settings*, não tem a capacidade de realizar qualquer acção, mas que graças à sua arquitectura única pode ser convertida numa qualquer ferramenta que seja necessária apenas no domínio digital, visto que não trabalham em analógico, e nesse mesmo domínio ser-lhe embutido qualquer programa.

É a extrema flexibilidade que as torna uma ferramenta apelativa para encarar variados problemas no ramo da electrónica. Tornando possível a sua conversão em arquitecturas de famílias distintas de micro controladores como AVR, que são RISC (*reduced instruction set computer*) *single chip* de 8 *bits*, ou da gama PIC, igualmente de 8 *bits*. É também comum e fazível desenhar um DSP *digital signal processor* ou um *driver* que seja possível customizar para um qualquer número de LEDs, não sendo a sua flexibilidade limitada pela complexidade do projecto [3].

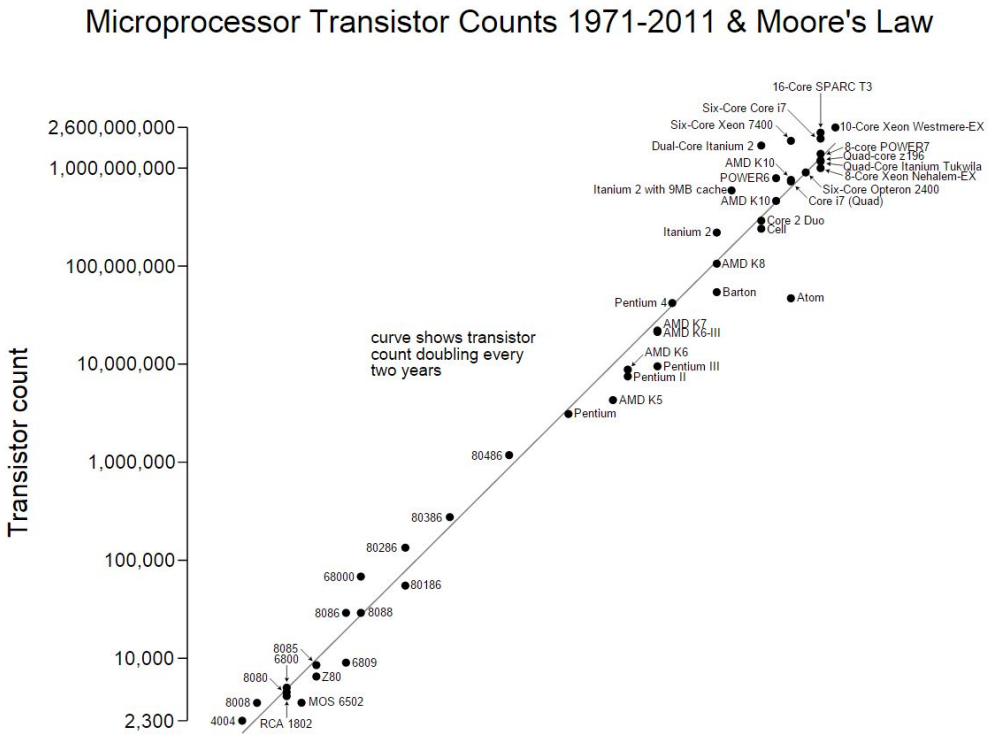

Figura 2.2: Crescimento exponencial do número de transístores conforme os anos [5]

2.1.1 *Sea of Gates*

Esta versatilidade é alcançada devido à sua estrutura maleável, que contém dentro de si milhares de pequenos elementos de lógica individuais, os CLB ou *configurable logic block*, sendo que o nome varia conforme o fornecedor. Têm a capacidade de implementar essencialmente qualquer função básica concebível e devido à existência de possivelmente milhares destes blocos, cabe assim ao utilizador a possibilidade de os configurar e moldar de forma a que possam desempenhar variadas funções digital independentemente da sua complexidade.

Estes CLB podem ser considerados mais que meros *gates* individuais, sendo blocos inteligentes e flexíveis no seu próprio direito. Encontram-se todos rodeados de uma matriz de linhas interligadas, a funcionar como conectores metálicos no interior do chip com a possibilidade de serem movidos de forma a reorganizar o *layout* interno da placa.

Como por exemplo, ligar de um CLB a outro e de seguida a um dos *pins* I/O (*input / output*) ou vice-versa, de forma a implementar a função desejada na FPGA. A este conjunto é frequentemente dada a denominação de "*sea of gates*", como exemplificado na imagem 2.3, devido a serem precisamente isso, um grande número de *gates* a flutuarem num "oceano" de lógica interligada, apesar de tal denominação ser um pouco redutora [6].

Hipoteticamente, a FPGA ou *configurable logic chip* mais poderoso imaginável poderia conter até milhões de portas NAND e uma rede infinita de linhas que ligassem cada peça entre si no seu *sea of gates*, pois como a teoria lógica básica dita, com um número suficiente destas é possível gerar qualquer estrutura. Desde um micro controlador a centenas destes, ou até qualquer aplicação de processamento de sinais digitais ou arquitectura desejada, a partir destas portas mais simples.

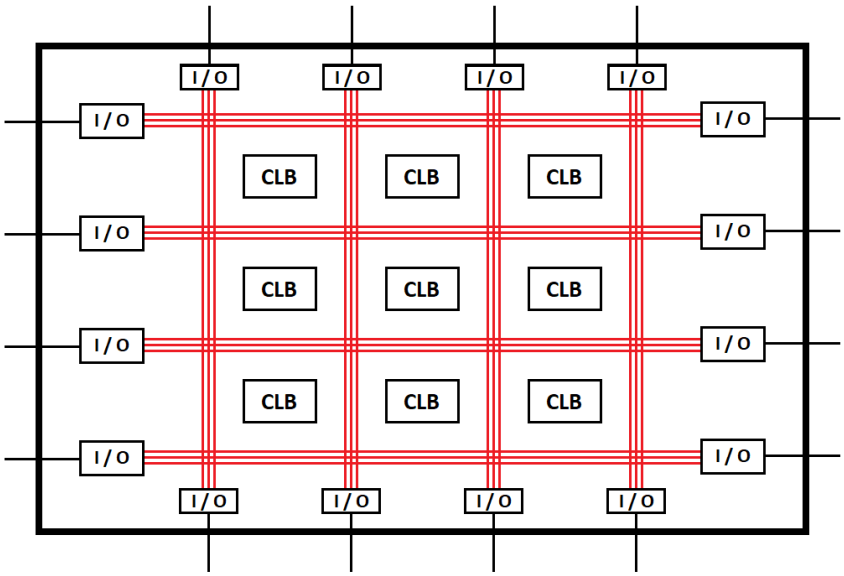

Figura 2.3: Estrutura interna de uma FPGA

Tais chips, no entanto, já foram concebidos e são denominados *gate arrays*, mas têm vindo a cair em desuso. Partilham de um conceito semelhante às FPGA (que tomaram o seu lugar graças aos seus CLB mais avançados) pois em teoria seria também possível ter todas as portas NAND configuráveis dentro deles, mas teriam de lidar com a falta de espaço, devido ao elevado grau de complexidade.

Não é então exequível ter simultaneamente uma rede infinita de todos estes fios a cruzarem-se juntamente com um número ilimitado de *gates*, e também fazer com que a combinação de ambos fosse configurável e possível de orientar em qualquer das direcções dentro do *chip*. Rapidamente se transformaria num impasse de *routing* (tal como num PCB) dentro da FPGA, e no final fazer com que a versatilidade das placas se tornasse um impedimento [3].

2.1.2 *Configurable Logic Blocks*

A arquitectura das FPGA modernas é então uma optimização de CLBs mais complexos que realmente contêm portas individuais, mas também elementos como *flip-flops* ou *lookup tables* (LUT) visto na imagem 2.4.

Quanto mais elementos se encontram rodeados por um número limitado de linhas interligadas desta forma, mais a arquitectura se torna um equilíbrio do *trade-off* entre a operação lógica que é possível desempenhar para o resultado necessário com os recursos disponíveis e as ligações possíveis de efectuar com a quantidade existente de fios, de forma a conseguir fazer chegar os sinais a todos os *pins* I/O na fronteira dos elementos lógicos internos [6].

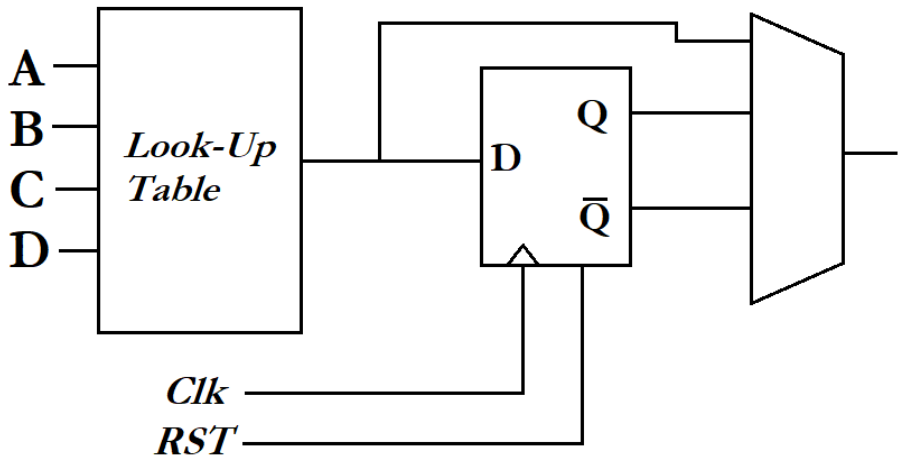

Figura 2.4: Estrutura interna de um CLB

Geralmente apresentam uma complexidade maior, mas esta aproximação genérica do interior de um CLB contém os elementos essenciais: o *flip-flop* e a *lookup table* básica com o número de inputs a depender da FPGA seleccionada, e dentro da LUT encontrar-se-ão todos os fusíveis de configuração que podem ser alterados para implementar uma determinada função. Os *flip-flops* no seu interior podem ser usados de diversas formas e encontram-se ou ligados a linhas internas de *clock* ou a sinais que podem vir do exterior da placa.

Além de entradas dedicadas a *clock* e a *reset*, estes permitem a escolha de *output* Q ou não Q que depois irá ser adicionado à saída da LUT, sendo possível escolher qual das linhas será a saída final do CLB.

O conjunto final no total apresenta assim uma série de inputs versáteis, portas lógicas configuráveis e uma arquitectura *flip-flop* básica para executar alguma *latching logic* dentro da já existente *clocked logic*, que resulta num poder cumulativo considerável sabendo o número elevado de CLB existentes dentro da FPGA.

2.1.3 *Input / Output*

Os blocos de I/O em toda a periferia do chip que se irão ligar depois aos *pins* individuais I/O são também possuidores de alguma complexidade por si só e não apenas um *buffer*, que efectuaria a saída do sinal para o *pin*, podem também ser programados para desempenhar variadas funções.

Tal como um micro controlador, o utilizador pode programa-los para serem *inputs*, *outputs* e podem ser até transformados em *tristate inputs*.

Podem ser convertidos em *drivers* de pares não só *single ended* como diferenciais, com a possibilidade de implementar diferentes *standards* de tensão de carga em diferentes *pins*, contendo ou *flip-flops* ou *latches* de forma a correr memória DDR e operações semelhantes nesse nível de complexidade.

Tudo isto contido nos blocos I/O que estão ligados ao *sea of gates*, a matriz de fios interna [6].

2.1.4 Blocos DSP

Embora um CLB seja capaz de executar operações aritméticas e armazenamento de dados, seria muito lento e utiliza uma enorme quantidade de recursos para uma função possivelmente mais simples pois um CLB foi projectado para ser relativamente versátil, de forma a executar diferentes tipos de funções.

Enquanto a procura por FPGA foi aumentando nesta última década, iterações mais modernas da tecnologia oferecem blocos incorporados para executar funções, algumas das quais são blocos de memória e blocos DSP, *digital signal processing*. A introdução destes blocos incorporados reduzem claramente a utilização da área dentro dos recursos disponíveis, o consumo de energia e, como consequência, resulta em melhor desempenho [7].

Embora a estrutura interna dos blocos DSP e os seus detalhes embutidos sejam diferentes, dependendo na versão do fabricante e do dispositivo, a maioria das arquitecturas dos blocos DSP não apresenta ter diferenças aparentes.

Um bloco DSP consiste assim num somador e multiplicador seguido por um acumulador. O objectivo da inclusão de blocos DSP visa aprimorar o desempenho das operações aritméticas a realizar.

Além disso, existem ligações específicas em cada bloco DSP que podem ser usadas para conectar vários DSP blocos uns aos outros que, por sua vez, podem ser usados para implementar um filtro FIR eficiente [7].

2.1.5 Funcionamento

Unindo todos estes elementos, existe uma arquitectura poderosa e versátil dentro do chip capaz de desempenhar uma qualquer função concebível, mas que ao ser inicializada continua a não "saber" o que fazer, pois é tipicamente um dispositivo volátil e não contém forma de armazenar a sua configuração interna, independentemente de ser como ela foi programada para estar estruturada ou do programa que lhe foi incutido pelo utilizador a inicializar.

Assim que deixa de estar ligada, a FPGA "esquece" e volta a ser o chip que veio da fábrica, com apenas o *sea of gates* à espera de novas instruções. Para contornar esta volatilidade, as FPGA têm embutido um pequeno bloco lógico que liga externamente a uma *flash memory* para configuração.

Portanto a utilidade da FPGA depende de dispositivos periféricos que lhe dão um propósito, neste caso não necessita de ser mais do que uma pequena memória de 4 a 8 Megabytes a correr paralelamente, que contenha em si toda a informação necessária para organizar os fusíveis interinos da FPGA. "Fusíveis" estes que não são na verdade fusíveis físicos, mas transístores ou *latches* que são ligados e armazenam a função nesse *bit* em particular, ditando todas as conexões a serem formadas dentro do *sea of gates*.

Assim quando é inicializada a FPGA, com auxílio da memória externa, irá carregar a configuração necessária para o seu funcionamento, através do bloco lógico de configuração que sabe automaticamente como carregar a informação da *flash memory* e programar todos os seus fusíveis. Esta acção pode demorar alguns segundos ou mais, sendo que as FPGA não são dispositivos instantâneos devido aos segundos necessários para se dar a inicialização.

2.1.6 Vantagens

As desvantagens da tecnologia são o preço inicial mais elevado, os requisitos de consumo energéticos mais exigentes que outros dispositivos semelhantes, a sua volatilidade inerente e a complexidade das suas ferramentas juntamente com o seu uso num todo.

Estas restrições iniciais em junção com a dificuldade que é simplesmente comparar duas FPGA – torna a sua aprendizagem algo que afugenta iniciantes.

No entanto, a mera noção de existir uma falta de limites nas implementações de projectos torna esta tecnologia um investimento lucrativo, visto que no domínio digital e a esta escala, não há constrangimentos. A FPGA apresenta-se então como um chip imponente mas com uma curva de aprendizagem abrupta, que pode ser alterado de forma a desempenhar as mais variadas funções sem restrições de complexidade, desde um micro controlador a uma *drive* personalizada para gerir um cubo de LED.

2.1.6.1 Velocidade

Algumas das mais básicas FPGAs contêm blocos I/O e funcionalidade em série que lhes permitem atingir velocidades na ordem dos Gigabytes por segundo, e transceptores embutidos que conseguem igualmente fazer descodificação na mesma ordem, com todos os componentes a prezar para a velocidade da arquitectura final.

Comparando com, por exemplo, um micro controlador, até um processador rápido como os que se encontram nas Raspberry Pi, é capaz de conseguir fazer alternar o I/O a 100 MHz ou a frequências semelhantes, mas irá sempre deparar com a problemática do *bottlenecking* do CPU, tornando qualquer processamento sério impossível de realizar num sinal de input nessa mesma gama, enquanto que numa FPGA essa problemática é inexistente. Torna-se simplesmente uma situação de alocação de recursos, neste caso dedicar CLB para atacar esse mesmo problema, o que se traduz numa vantagem enorme de velocidade em função de paralelismo, ao realizar apenas uma função específica num único sinal I/O desejado.

Portanto se existe um sinal a entrar e a intenção é trabalhar nele com rápido processamento (no que será efectivamente tempo real) e fazer a informação sair por outro *pin*, é apenas necessário dedicar uma parte da FPGA para o fazer.

Por exemplo, codificadores de quadratura são populares como um mecanismo para *feedback* de posicionamento para os motores usados em impressoras 3D. Codificadores baseados em *software* podem provar ser problemáticos devido a limitações óbvias na velocidade, causando falhas de contagem quando o processador está ocupado a executar outro código.

Com os codificadores de quadratura instanciados na FPGA e com um *clock* de 100MHz implementado em *counters* de 32 bits, a possibilidade de falhar alguma contagem num sistema real físico é aproximadamente nula. Estando cada codificador implementado usando blocos da FPGA em separado, a perda de performance devido ao facto de existirem vários em paralelo é inexistente, instalar mais significa apenas uma alocação de mais recursos internos da FPGA.

2.1.6.2 *Field programmable*

Tal como o nome indica, a diferença que isso traduz de um micro controlador que pode ser reprogramado ou *reflashed* na hora é de que não existe um processador fixo até este ser programado pelo utilizador, portanto tudo pode mudar no seu design. É então possível uma adaptação a qualquer problema.

Tendo, por exemplo, programado a placa para servir de micro controlador e chegando à conclusão de que é necessário um aumento de potência para alimentar o chip e uma maior capacidade de processamento, visto que o processador corre dentro da FPGA, se houver recursos suficientes em excesso que não estejam em uso, é possível construir uma solução adequada. Quer ela seja implementar um filtro FIR entre dois pins I/O ou um

controlador PID, são mais poderosas nesse aspecto do que um micro controlador pois não estão restringidas a um único processador.

2.1.6.3 Permitir paralelismo

Existe a possibilidade de investir vastos recursos com um *pay off* satisfatório em várias tarefas em simultâneo, visto que, ao contrário dos competidores, não requer que os processos passem todos sequencialmente por um único núcleo de processamento restringindo sua a capacidade de *output*.

Tomando um micro controlador básico como exemplo, este é efectivamente um *bottleneck* como referido na sub-subsecção 2.1.6.1, visto que tudo tem de passar pelo CPU sequencialmente e as linhas individuais de código são assim executadas uma por uma. Para que haja qualquer tipo de semelhança a processamento em tempo real é preciso que tudo este processo seja feito bastante rápido, e mesmo que se tenha um sistema operativo em tempo real, esse resultado não será verdadeiro tempo real.

Não é possível processar as acções de 10 *pins* de um lado do chip no exacto instante em que se faz o mesmo do lado oposto ao mesmo número de *pins* enquanto se vai alternando os outputs. Tal acção com uma estrutura típica simplesmente não é exequível sem albergar uma arquitectura que permita várias acções em paralelo. Enquanto que com recursos CLB suficientes esta acção ou mais poderia ser tomada sem detrimento à performance do dispositivo e hipoteticamente, multiplicada as vezes que fossem desejadas.

Tomando, por exemplo, um conversor analógico digital, se estiverem disponíveis 50 canais de dados das amostras do ADC a entrar através de um chip ADC externo (visto que os *pins* são digitais e não conseguem processar sinais contínuos) e a intenção fosse processar toda essa informação em simultâneo, uma solução possível que não requeresse paralelismo seria deixar uma amostra dos 50 canais *on hold* e depois esperar que o processador os despache um por um, o que resulta num *bottleneck* grave conforme a complexidade da amostra. Numa FPGA bastaria alocar um grupo de CLB para trabalharem exclusivamente num canal e depois copiar esses recursos as vezes que forem necessárias. É esta a enorme capacidade de executar funções em paralelo da tecnologia FPGA, a possibilidade de processamento simultâneo em larga escala.

2.1.6.4 Elevado número de *pins*

Por fim, contêm um número elevado de *pins* I/O, sendo um dos focos desta tecnologia que recebeu um "empurrão" no sentido de albergar o número máximo de *pins* possíveis e desenhados com foco em aplicações que façam uso do número elevado de saídas. Sendo, inclusive, difícil encontrar um exemplar decente de uma FPGA – estando a sua qualidade em correlação directa com o número de CLB que efectivamente contém – com um número reduzido destes *pins* [8].

2.1.7 Alternativas

A figura 2.5 mostra como as tecnologias CPU, GPU e ASIC diferem, desde o CPU ser altamente flexível com rápida reprogramação mas com pouca eficiência entre poder de computação por *watt*, aos ASIC serem altamente eficientes nesta mesma relação mas sem qualquer flexibilidade, não tendo possibilidade de serem reprogramados com o requerimento de ciclos de desenvolvimento com a duração de anos.

Esta comparação demonstra a superioridade da FPGA, eficiente no poder de computação por *watt*, possibilidade de ser reprogramável e relativamente eficiente em termos de consumo de energia.

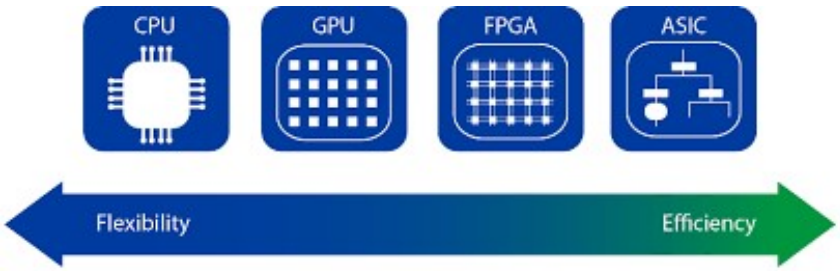

Figura 2.5: Comparação da flexibilidade e eficiência nas diferentes arquitecturas [4]

2.1.7.1 Futuro

Nos últimos anos, *hyperscale data centers* e provedores de serviços têm-se debruçado sobre a problemática da gestão de informação a larga escala e tentado arranjar soluções. Em 2010, a equipa do Microsoft Azure começou a investir na tecnologia FPGA e em 2012, foi criado um protótipo com 60 placas FPGA com o propósito de acelerar projectos como classificações de índice das buscas do motor Bing, e os resultados promissores levaram à inserção de 1600 FPGA na *network* de produção no ano seguinte [4].

Com isto houve a realização de que FPGA seriam vitais em acelerar as tarefas dos projectos com o *trade-off* perfeito entre custo e performance. Em 2014 foi introduzida uma nova placa em que, com base em experiências anteriores dos primeiros modelos e alguns problemas de *network*, foi decidido colocá-la *inline* (entre a *network* e o CPU) de forma a permitir uma aceleração nos seus servidores na parte de *networking*. Desde o final de 2015 em diante foi colocada uma FPGA em cada servidor, mesmo antes de terem o *software* preparado para as receber, o que só aconteceu no ano seguinte [4].

Outro marco importante na história desta tecnologia foi a aquisição recente (2015) da Altera, um dos dois maiores fabricante das placas na indústria, por parte da Intel, demonstrando um interesse e investimento maior na tecnologia como um componente-chave na infraestrutura de servidores. Desde então que a Intel tem estado a trabalhar em inserir tecnologia FPGA nos servidores, equipando os processadores de micro arquitectura Skylake com FPGA embutidas na *package* do CPU.

Estes casos demonstram o potencial e o ímpeto que a tecnologia tem vindo a ganhar como *motor de aceleração* dentro de *clouds* como de provedores de serviço, com empresas como Microsoft Azure, Amazon Web Services entre outros a oferecerem *FPGA-as-a-Service* aos seus clientes de forma a usufruírem da mesma aceleração que dispõem internamente.

Com a FPGA, a metodologia de desenvolvimento de *software* é trazida para o *hardware*, exigindo assim ao utilizador que programe, construa, faça o *download* e por fim teste, para que no fim possa repetir o processo *ad nauseam* e não estando restrito com problemas de processamento e aumentando o seu tempo de vida no mercado.

As *Field Programmable Gate Array* são assim uma "plataforma de computação reconfigurável" a providenciar aceleração flexível pela eficiência adequada, que apesar das exigências do mercado e velocidade das inovações tecnológicas, demonstra ser fiável a médio/longo prazo.

2.2 *Hardware Description Languages*

Engenheiros e *designers* de *hardware* electrónico descrevem o comportamento e a estrutura de projectos de sistemas ou circuitos usando linguagens de descrição de *hardware* (HDL - *hardware description languages*) - linguagens de programação especializadas normalmente denominadas por VHDL, Verilog e SystemVerilog.

Estas diferem das linguagens de programação de *software* porque incluem uma forma de descrever o tempo de propagação e a força do sinal. Hoje em dia, seria impossível projectar um sistema complexo num circuito integrado ou *chip* (SoC, *System on a Chip*) para um dispositivo móvel ou qualquer outro produto com forte componente electrónica sem recorrer a uma HDL. [9]

Ao contrário de linguagens de programação em que o micro controlador executaria o código sequencialmente, Verilog é uma linguagem descritiva, em que todas as linhas serão avaliadas e executadas simultaneamente, logo a ordem de escrita não é relevante.[10].

Cada uma das três HDLs apresenta as suas características e estilo distinto, com VHDL e Verilog a implementar abstracções a nível de transferência de registo (RTL - *register-transfer-level*). Quando foram introduzidas pela primeira vez no final dos anos 80, eram consideradas tecnologias inovadoras visto permitirem aos engenheiros trabalhar a um nível mais alto de abstracção com simuladores RTL, pois antes da sua chegada os projectos eram unicamente simulados ao nível de esquemático ou de *gates*.

Com o Verilog e o VHDL, a funcionalidade desejada pode ser representada como um programa de *software*. Em seguida, o modelo é simulado para confirmar que o *design* funcionará como pretendido, com quaisquer problemas a poderem ser corrigidos no modelo e a simulação irá verificar a correcção [9].

VHDL e Verilog são consideradas linguagens de *design* digital de uso geral, enquanto que o SystemVerilog representa uma versão aprimorada do Verilog. VHDL é uma linguagem rica e fortemente *typed*, determinística e mais detalhada que a Verilog. Como resultado, os projectos escritos em VHDL são considerados auto-documentados.

A sintaxe não é do tipo C e ao trabalhar em VHDL é necessário executar codificação extra para converter de um tipo de dados noutro. O VHDL geralmente captura erros perdidos pelo Verilog, com ênfase em semânticas inequívocas e permite a portabilidade entre ferramentas.

No início dos anos 90, a indústria da electrónica foi forçada a enfrentar as "*language wars*", onde facções concorrentes no campo VHDL ou Verilog competiam pelos projectos e espaço no *desktop* dos engenheiro. Estas duas linguagem sobreviveram e agora coexistem, muitas vezes seguindo o mesmo pensamento de *design*, sendo que todas as três são *IEEE industry standards* [9].

2.3 Arduino MKR Vidor 4000

A primeira placa de desenvolvimento Arduino foi introduzida em 2005 e era baseada num simples micro controlador de 8 *bits*, o Atmel AVR, tendo muitas das subsequentes gerações sido baseadas em membros desta família de micro controladores. Devido ao seu baixo custo e à acessibilidade do Arduino IDE, *Integrated Development Environment*, as placas cresceram em popularidade tanto como plataforma de desenvolvimento de IoT como para projectos de engenharia a um nível "profissional" como académico.

A placa original Arduino tem também a distinção de ser o primeiro projecto de *hardware open source* a desfrutar de sucesso e reconhecimento universal. É um favorito para expor estudantes ao mundo de programação embutida, tendo colaboradores por todo o mundo fora a contribuir com criações suas para a série Arduino, com aplicações desde *hobbies*, casas inteligentes e agricultura ou até veículos autónomos.

Conforme estes avanços e contribuições dos utilizadores têm aparecido, usando a arquitectura Arduino original com aplicações cada vez mais complexas, foram também deparados impasses na tecnologia e problemas de performance, devido em parte à sua simples estrutura 8-*bit* do micro controlador original.

2.3.1 Adaptação

O cerne do problema encontrava-se na implementação do *software* de muitos periféricos com o intuito de funcionar em tempo real, em especial aplicações periféricas dependentes de tempo como temporizadores de alta velocidade, codificadores em quadratura e *outputs* de *pulse width modulation* (PWM), de alta frequência.

Uma forma de encarar estes problemas é usando micro controladores mais rápidos e com mais poder de processamento, mas tal solução é um beco sem saída pois há um limite para o que *software* pode fazer e resolver num ambiente de tempo real – certas funções de alta velocidade e tempo real necessitam de ser implementadas em *hardware*, como dito na sub-subsecção 2.1.6.3.

Como referido anteriormente, *firmware* a correr num micro controlador ou microprocessador nunca irá prezar pela velocidade e pode até provar ser bastante lento criando cenários de extremo *bottleneck*. Foi aqui que as FPGA entraram com a oportunidade de uma opção programável de resolver a problemática da baixa velocidade, tempo real e incongruências de design integrado a um nível de *hardware*. No entanto, retirar o máximo uso de uma FPGA tradicionalmente requer embarcar numa curva de aprendizagem tecnológica que a maior parte dos designers escolhe não enveredar devido a constrangimentos de prazos e de custos acrescidos [11].

Esta curva regularmente envolve aprender linguagens novas como Verilog ou VHDL, fazer o *download* de vários pacotes de ferramentas de desenvolvimento, aprender a gerar *bitstreams* de uma FPGA e possivelmente investir uma quantidade significativa de dinheiro para uma placa de desenvolvimento nova para poder abordar esta tecnologia.

Devido a todos estes obstáculos muitos designers evitam usar FPGAs, mesmo quando elas provam invariavelmente ser a ferramenta certa para o trabalho e no geral bastante úteis na área do design embutido.

Vários produtores tentaram reduzir os desafios iniciais ao uso da FPGA ao oferecer placas para principiantes, mas a necessidade de aprender novas linguagens de forma a gerir o *hardware* e de familiarizar com outras ferramentas de desenvolvimento continua a apresentar-se como um desafio considerável que tem impossibilitado um uso mais generalizado da tecnologia.

Como forma de atacar esta problemática, a Arduino apresentou uma nova placa de desenvolvimento *low-cost* chamada "Arduino MKR Vidor 4000", que representa uma abordagem única ao uso de FPGAs em sistemas de design embutidos.

O Arduino MKR Vidor 4000 contém formas de programar a FPGA que se encomtram integradas na já popular Arduino IDE, o *software* usado para escrever código e fazer o respectivo *upload* para a placa física. Desta forma, será possível em teoria obter uma *performance* expectável ao nível das FPGAs usando as extensões de bibliotecas disponibilizadas pelo Arduino. Estas ferramentas permitem aos *developers* de sistemas embutidos aceder rapidamente aos benefícios a partir de um *sketch* do Arduino, enquanto designers mais avançados poderão adoptar modelos de FPGA mais complexos conforme o domínio com esta tecnologia cresce [11].

2.3.2 Arduino e FPGA

O desafio apresentado era o de unir os vastos recursos e capacidades da FPGA à simplicidade da plataforma Arduino, de maneira a tornar ambas as vantagens disponíveis aos *developers* sem comprometer qualquer uma. A solução encontrada, de um ponto de vista de *hardware*, a esta problemática e a apresentada neste modelo foi de adicionar uma FPGA à arquitectura Arduino de forma a que a FPGA estenda os recursos periféricos do micro controlador do Arduino através das bibliotecas deste. Estas bibliotecas são familiares aos utilizadores já acostumados aos modelos mais simples e são usadas frequentemente para fazer uso dos periféricos já nativos da marca, assim como quando se pretende adicionar placas periféricas aos *expansion headers* deste.

A placa resultante e em questão, MKR Vidor 4000, combina a capacidade de processamento de um Microchip Technology SAMD21, micro controlador de baixa potência baseado num *processor core* 32-bit Arm® Cortex®-M0+, com uma FPGA Intel (anteriormente Altera) Cyclone 10 10CL016.

Figura 2.6: A Placa Arduino MKR Vidor 4000 [12]. **1** - *USB*; **2** – *I2C*; **3** – *Conector LiPo*; **4** – *Reset*; **5** – *SAMD21*; **6** – *Memória da FPGA*; **7** – *FPGA*; **8** – *MicroHDMI*; **9** – *NINA WiFi*; **10** – *Conector de Câmara MIPI*; **11** – *MiniPCI Express*.

A FPGA contém 15408 blocos de lógica configuráveis, 516096 bits de RAM embutida e 56 18x18 bit *hardware multipliers* para implementar processamento de sinais digitais de alta velocidade. Cada um dos *pins* I/O consegue alternar a uma velocidade maior do que 150 MHz, o que é consideravelmente mais rápido que as capacidades I/O de um micro controlador SAMD21 *on-board*. Em resumo, a FPGA embutida contribui com bastante poder de processamento e alternância I/O à placa Arduino MKR Vidor 4000.

Esta placa é um exemplo de uma série Maker Arduino com um novo MKR *form factor* que não coincide com modelos anteriores, visto que esta série MKR da Arduino foi desenvolvida com o propósito de ser usada tanto por principiantes como por profissionais,

com o intuito de desenvolver rápidos protótipos para sistemas, e surge após ter sido observado o uso contínuo destas placas de desenvolvimento em numerosos projectos de engenharia desde a fase de protótipo até à produção.

2.3.3 Composição

A placa Arduino MKR Vidor 4000 inclui um número de periféricos, entre os quais alguns que não estão geralmente associados com os modelos prévios e de natureza mais simples. Existe a 28-*pin* MKR pin header interface, semelhante em conceito, mas fisicamente diferente da original *shield header* da Arduino. Os *pins* I/O nesta interface podem ser controlados tanto pelo micro controlador SAMD21 ou pela FPGA Cyclone 10. Em adição à estandardizada MKR *pin header interface*, o resto dos periféricos da placa inclui:

- Micro conector USB;
- Micro conector de vídeo HDMI em *output*;
- Conector de câmara de vídeo MIPI compatível com módulos de câmara da Raspberry Pi;
- Interface RF de *Wi-Fi* e BLE implementada com um módulo transceptor U-BLOX NINA-W102 RF;
- Conector MiniPCIe com um máximo de 25 *pins* programáveis.

De notar que a placa de desenvolvimento Arduino MKR Vidor 4000 não tem uma verdadeira entrada PCIe, *peripheral component interconnect express*, ela meramente adapta o conector associado a uma entrada MiniPCIe, "danificando" assim muitos dos *pins* I/O da FPGA Cyclone 10 no conector de canto MiniPCIe. Trata-se de um conector barato e de alto volume, com uso recorrente em milhões de *motherboards* de PC, que facilita a ligação da placa a um sistema maior ou a uma *expansion board* de *pins* I/O.

Apesar de conter um número elevado de conectores I/O, a placa tem dimensões reduzidas: 83 mm por 25 mm. A sua taxa de consumo está na ordem dos 100 mA, portanto os requisitos energéticos, tal como o seu tamanho físico, permitem a sua inclusão num leque grande de projectos com sistemas embutidos. A MKR Vidor 4000 inclui um conector usado para ligações directas de 3.7 V de bateria LiPo (*lithium polymer*), no entanto a taxa de consumo de corrente irá variar conforme o que foi implementado na FPGA e a velocidade à qual o *hardware* instanciado irá correr, o que complica os cálculos para a capacidade da bateria e irá requerer um grau de experimentação de forma a acertar com a bateria ideal.

O canal de saída de vídeo da placa é um *output* HDMI verdadeiro gerado pelo Intel Cyclone 10 FPGA, tendo sido incluído na Arduino IDE uma biblioteca gráfica (Adafruit GFX) de modo a permitir ao micro controlador SAMD21 gerar gráficos de acordo com o que é pedido pelo *software*.

Foram também adicionados vários blocos de IP de *hardware* à IDE através de duas bibliotecas principais desenhadas especificamente para a placa MKR Vidor 4000: VidorPeripherals e VidorGraphics. Estes blocos de *hardware* são automaticamente construídos na FPGA no instante em que a biblioteca periférica é incluída no *sketch* do Arduino. Estes blocos incluem: I2C *ports*, SPI ports, UARTs, *pulse width modulation controllers* de alta frequência, temporizadores de alta-velocidade, descodificadores de quadratura de alta velocidade e um controlador Neopixel RGB Smart LED da Adafruit.

A biblioteca de *software* da placa contém também código que permite captação de vídeo através da entrada MIPI, possibilitando a captura de imagens de 640 x 480 pixeis por uma câmara externa. As imagens RGB de 24 bit da câmara são armazenadas na SDRAM de 8 megabytes da placa, que se encontra directamente ligada à FPGA.

É também possível usar uma das interfaces I2C para aceder aos controlos da câmara externa. Alguns destes periféricos, com o I2C e SPI, já existiam como blocos de *software* para placas Arduino anteriores, sendo que a diferença entre estas versões e os blocos IP implementados na FPGA da placa MKR Vidor 4000 é a sua optimização. No caso destes dois últimos, a sua instalação é implementada com diferentes blocos de *hardware* dentro da FPGA da placa, e como consequência, não existe perda de performance quando estes *ports* estão a ser instanciados.

2.3.4 Uso de FPGA de forma Arduino

O intuito desta placa foi tornar o uso da tecnologia mais acessível ao implementar estes blocos periféricos como *hardware* da própria FPGA, tornando-os disponíveis usando o mesmo mecanismo de bibliotecas já conhecido dos utilizadores. Enveredar pelo Arduino IDE, em teoria não deveria existir nenhum requerimento de uma nova linguagem de *hardware* como VHDL ou Verilog para aprender, portanto do ponto de vista do programador, periféricos com base na FPGA têm o aspecto de qualquer outro objecto de C++.

Esta solução de fazer o *hardware* de uma FPGA parecer blocos periféricos foi conseguida através de uma *stack* de *software* que se estende para lá da configuração original do Arduino IDE.

Este *stack* da placa cria uma ligação entre *sketches* de alto nível, escritos na variante Arduino da linguagem C++, e o *hardware* FPGA da placa usando *remote procedure calls* enviados para uma caixa de correio instanciada na FPGA e que está por sua vez ligada ao micro controlador SAMD21 através de um conector JTAG.

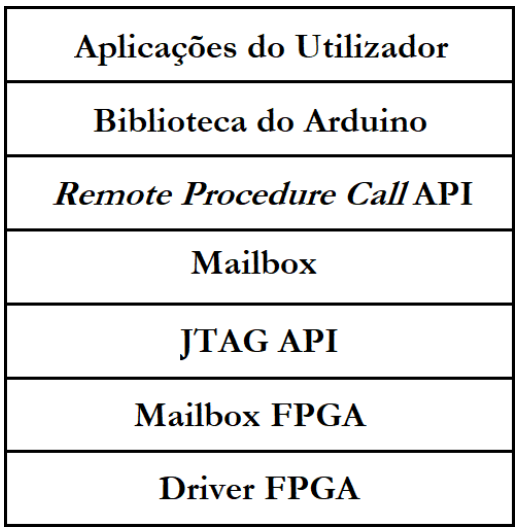

Figura 2.7: O *software stack* da placa MKR Vidor 4000 [11]

Sendo que as duas últimas camadas na figura 2.7 são executadas já na FPGA, este *software stack* usa então as chamadas de procedimento remotas transmitidas através de uma API JTAG para contactar a FPGA Intel Cyclone 10, que tem já um driver Arduino instanciado no *hardware* programável da FPGA. Com esta configuração, acesso aos periféricos com base na FPGA torna-se apenas uma questão de incluir as *calls* correctas para as bibliotecas Vidor correspondentes.

Embora esta forma de acesso a blocos IP periféricos implementados com *hardware* programável seja seguramente uma forma válida de melhorar o desempenho de um sistema através do uso de um FPGA, a *performance* resultante desta configuração é apenas uma demonstração das capacidades desta tecnologia. No entanto, o "empurrão" dado aos FPGAs por uma empresa como a Arduino significa o apoio de uma base extensa de utilizadores que irá desenvolver ainda mais blocos periféricos e fazer crescer ainda mais esta comunidade *open source*.

Só não é apenas uma placa FPGA "multiusos" pois ainda não beneficia de muito suporte para programação directa na Cyclone 10 usando código HDL.

Em suma, enquanto que os FPGAs oferecem uma melhor performance para várias funções de processamento em comparação com um micro controlador ou micro processador, vêm com uma curva de aprendizagem muito mais acrescida. A placa Arduino MKR Vidor 4000 e a acompanhante Arduino IDE tentam baixar então substancialmente o tempo necessário para esta fase inicial de aprendizagem, apresentando-se como a ferramenta para designers que planeiam desenvolver aplicações avançadas usando uma FPGA, mas que não estão familiarizados com a tecnologia.

O desafio mantém-se, foi apenas tornado mais acessível. [11]

2.4 Filtros

Quando se trata de processamento de sinais, será sempre necessário extrair a parte útil de um sinal e remover a indesejada, tal ato é feito maioritariamente através de uma acção de filtragem pelos componentes activos num filtro.

Neste processo, o sinal desejado é seleccionado dentro da gama de frequência desejada, e sinais aleatórios tais como ruídos são retirados. Os filtros distinguem-se em dois tipos: analógico e digital.

2.4.1 Filtros Analógicos

Filtros analógicos seguem sinais do mesmo tipo ou que estejam em variação contínua, usando componentes electrónicos passivos (ou qualquer dispositivo discreto) como resistências, bobines ou condensadores, que são usados para permitir a passagens de certas frequências rejeitando outros sinais analógicos.

Estes filtros são dos principais componentes de processadores digitais de sinais, e servem maioritariamente para medir, filtrar ou comprimir ondas ou sinais físicos do mundo real. São também essenciais em várias aplicações como altifalantes, em que o seu uso permite separar os sinais de áudio antes do altifalante, em linhas de telefone para separar conversões, ou em receptores de rádio para dividir e seleccionar uma estação em particular.

2.4.2 Filtros Digitais

O filtro digital é um dos elementos mais importantes em aplicações de engenharia electrotécnica, tais como: comunicações, controlo, e sistemas de biomédica. Estes são usados para remover ou melhorar uma gama de frequência num sinal digital. Irá remover um sinal desnecessário e trabalhar neste (apenas os valores amostrados do sinal resultante) as desejadas operações numéricas, de forma a reduzir o ruído de fundo e suprimir o sinal interferente.

Existem vários tipos de filtros que são classificados com base em vários critérios, como linearidade linear ou não linear, variação no tempo ou invariante no tempo, analógico ou digital, activo ou passivo e assim por diante. Filtros digitais dividem-se em dois: FIR, *finite impulse response*, e IIR, *infinite impulse response*, assim classificados de acordo com a sua duração da resposta ao impulso. [13]

Os filtros são condicionadores de sinal e pela sua função permitem que passem componentes AC e bloqueiem componentes DC. O melhor exemplo do filtro é uma linha telefónica, que actua como um filtro ao limitar as frequências a uma gama significativamente menor do que a do ouvido humano.

Esta acção de filtragem é feita pela acção da transformada de Fourier. Se o processo de resolução do problema tiver como único requerimento aplicar uma operação matemática linear a um conjunto infinito de amostras de entrada equidistantes (como por exemplo um *data stream* amostrado), e fizermos questão que seja um sistema linear e invariante no tempo, irá sempre resultar num filtro linear.

Estes filtros são completamente caracterizados pela sua resposta ao impulso, isto é, se for apresentado ao filtro um único valor não negativo (um impulso), então a resposta produzida pelo filtro será uma resposta ao impulso. Se esta for de duração finita, trata-se de um FIR, caso contrário será um IIR. [14]

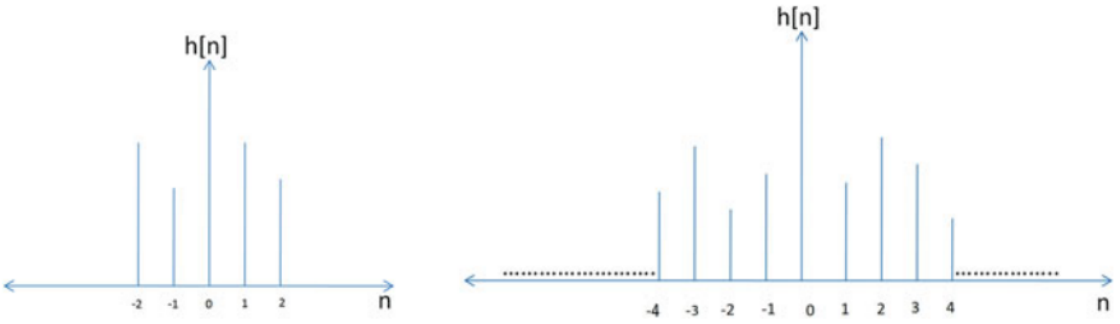

Figura 2.8: Resposta de impulsos finita e sequência de amostras infinitas [14]

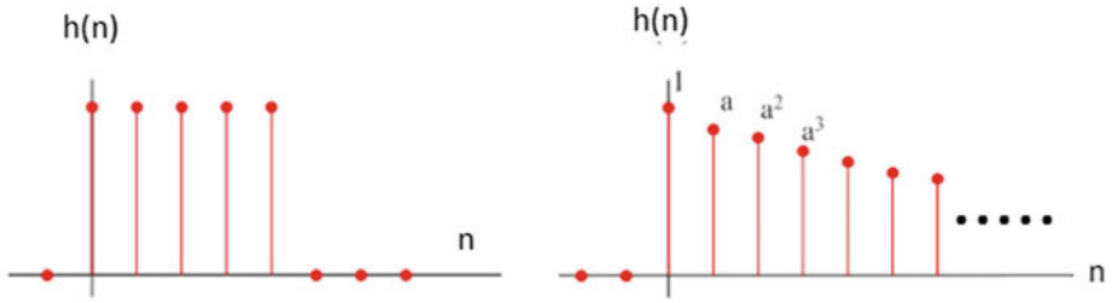

Figura 2.9: Resposta de um FIR numa janela rectangular e resposta infinita para a^n u(n) [14]

Filtros FIR são bastantes populares porque podem ser desenhados como sendo sempre estáveis e tendo exactamente a mesma fase linear, tendo sido usados extensivamente devido ao seu papel chave em várias aplicações de processamentos de sinais digitais (DSP – *digital signal processing*) [15]. Juntamente com os avanços conjuntos em tecnologia VLSI, *very large scale integration*, DSP tem-se tornado cada vez mais recorrente em aplicações modernas, levando ao surgimento e à subsequente exigência de filtros FIR de velocidades elevadas e um consumo cada vez mais reduzido.

2.4.3 Filtro FIR

Nesta secção, o foco será uma breve descrição do *background* teórico dos filtros FIR e as suas estruturas fundamentais. Um filtro FIR digital causal de comprimento N (ou ordem N - 1) pode ser caracterizado pela sua função de transferência, H(z), que é a sua transformada de z da resposta ao impulso do filtro digital [16].

$$H(z) = \frac{Y(z)}{X(z)} = \sum_{n=0}^{N-1} h(n)z^{-n} = \sum_{n=0}^{N-1} b_{n+1}z^{-n} \tag{2.1}$$

Podemos observar na equação 2.1 onde Y(z) e X(z) são transformadas de z dos sinais de *input* e *output* do filtro digital, respectivamente. E, *h(n)* e os coeficientes de *b* representam as respostas ao impulso e os coeficientes de filtro do filtro digital. Para um filtro FIR, os valores da resposta ao impulso e de coeficientes de filtro mantêm-se sempre inalterados, ao contrário de num filtro IIR em que terão sempre variação.

O filtro FIR pode também ser caracterizado pela seguinte equação 2.2 que nos demonstra a relação entre os sinais de *output* e *input* do filtro digital no domínio de tempo discreto [16].

$$y(n) = b_1x(n) + b_2x(n-1) + ... + b_Nx(n-N+1) \tag{2.2}$$

Onde *x(n)* e *y(n)* representam de novo os sinais respectivos de *input* e *output*. Comparado com os IIR, filtros FIR têm duas vantagens na sua estabilidade e características de fase linear, uma vez que são efectivamente sempre estáveis, devido aos pólos da função de transferência estarem sempre na origem do plano z, contidos assim dentro do círculo unitário.

Filtros FIR dispõe de uma facilidade inerente ao seu desenho devido a terem uma fase linear exacta, considerando que a função *h(n)* é simétrica ou assimétrica. Na literatura, quatro métodos generalistas, nomeadamente a optimização, *windowing*, amostragem de frequência e métodos numéricos são usados para projectar filtros FIR que contenham as características desejadas [16][17].

Usando uma ferramenta de *software* como o Matlab, um filtro FIR pode ser facilmente projectado e os seus coeficientes encontrados rapidamente.

Se permitirmos que a função *h(k)* represente a resposta deste filtro, então a operação matemática que descreve a forma como o filtro é aplicado a uma sequência de inputs *x(n)* trata-se de uma convolução, que pode ser definida pela equação 2.3.

$$x[n] \otimes h[n] \triangleq \sum_{k=-\infty}^{\infty} h[k]x[n-k] \tag{2.3}$$

Se *h(k)* for zero para, todo o k < 0 tal como k ≥ n, então *h(k)* pode ser usado para definir um filtro digital causal FIR. Esta distinção é importante pois apenas os filtros causais podem ser implementados em *hardware*, no entanto, para a implementação de um filtro FIR, a causalidade é uma boa garantia. A operação de um filtro FIR causal pode ser simplificada e representada pela equação 2.4.

$$y[n] = \sum_{k=0}^{N-1} h[k]x[n-k] \tag{2.4}$$

Onde *x(n)* é de novo a sequência de sinais de entrada, *h(k)* é a resposta deste filtro digital e *y(n)* é o *output* final. É esta a operação de um filtro digital, todo e qualquer filtro causal FIR terá de ter esta forma e é este o filtro a implementar na FPGA [13].

2.4.4 Estruturas Fundamentais de Filtros FIR

Filtros FIR digitais podem ser implementados recorrendo a três estruturas fundamentais conhecidas por: forma directa, forma directa transposta e forma simétrica. Além destas, podem ser implementadas recorrendo a outras estruturas tais como *cascade*, paralela, *lattice*, entre outras [13].

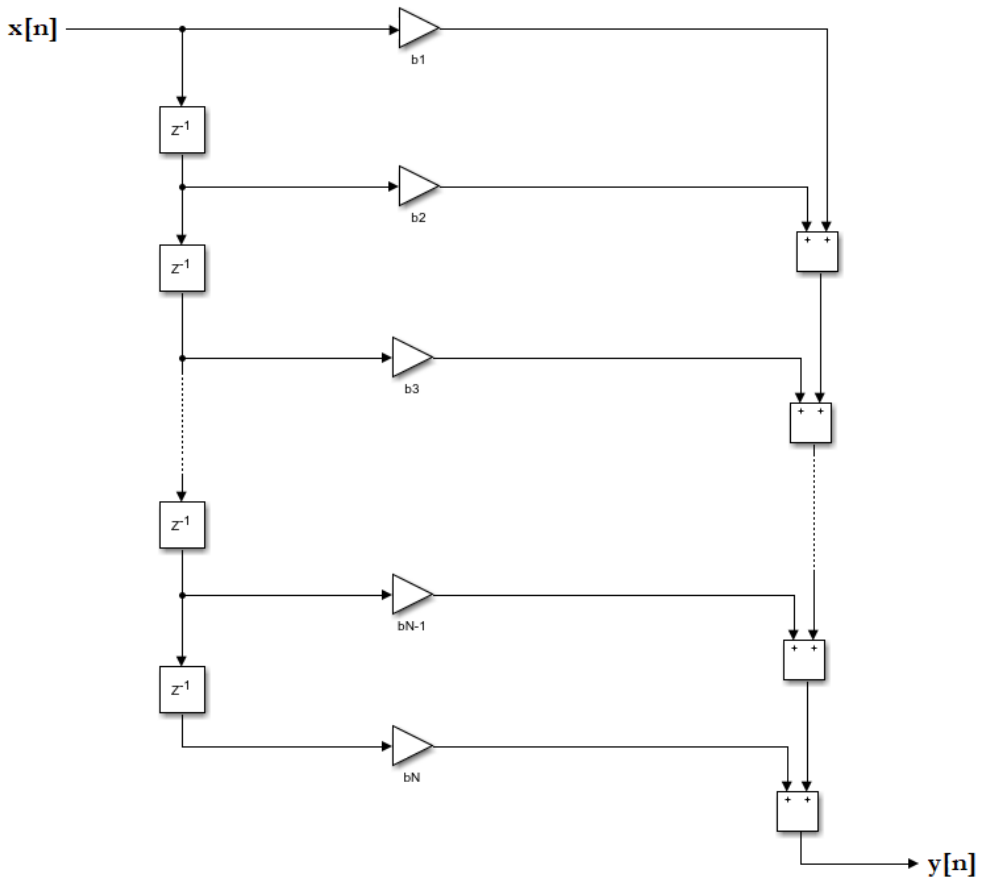

Figura 2.10: Estrutura de forma directa para um filtro FIR de comprimento N

Em todos os tipos de implementação de filtros digitais, são usados três elementos de circuitos: multiplicador (*multiplier*), somador (*adder*) e um atraso (*delay*). A estrutura do circuito para a implementação de um FIR de forma directa e comprimento N é demonstrado na figura 2.10, é assim chamada porque realiza o processo de convolução do filtro directamente.

Pode ser observado através da figura que o número total de elementos de atraso é igual à ordem do filtro (N-1), tornando-a uma *canonical type structure* [13]. Nesta estrutura, o sinal de *input* recebe primeiro um atraso por z-1 e é de seguida multiplicado pelos vários coeficientes *b*.

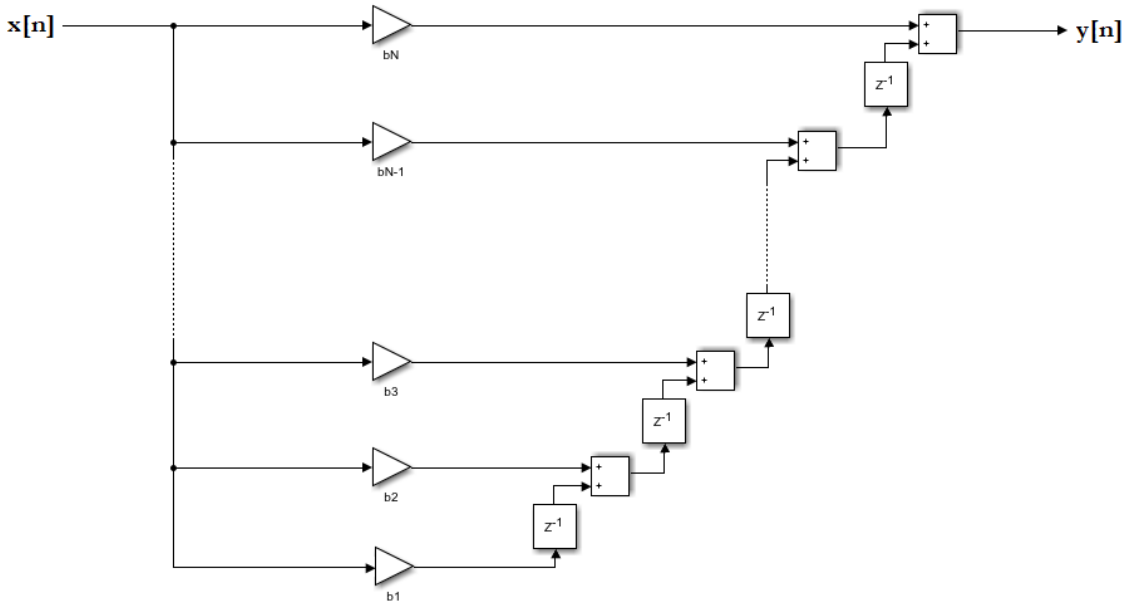

Figura 2.11: Estrutura de forma directa transposta para um filtro FIR de comprimento N

O circuito para a estrutura FIR de forma directa transposta e de comprimento N está representado na figura 2.11, tem a denominação de *transposta* pois ao contrário de na forma directa, o sinal de *input* é primeiro multiplicado pelos coeficientes *b* e só depois recebe o atraso por z-1. É também uma *canonical type structure*.

O circuito para a estrutura FIR de forma directa simétrica e de comprimento N está representado na figura 2.12. O número de multiplicadores nesta estrutura é metade das anteriores estruturas fundamentais nas imagens 2.11 e 2.10, e só pode ser usada quando os coeficientes do filtro a ser implementado forem simétricos.

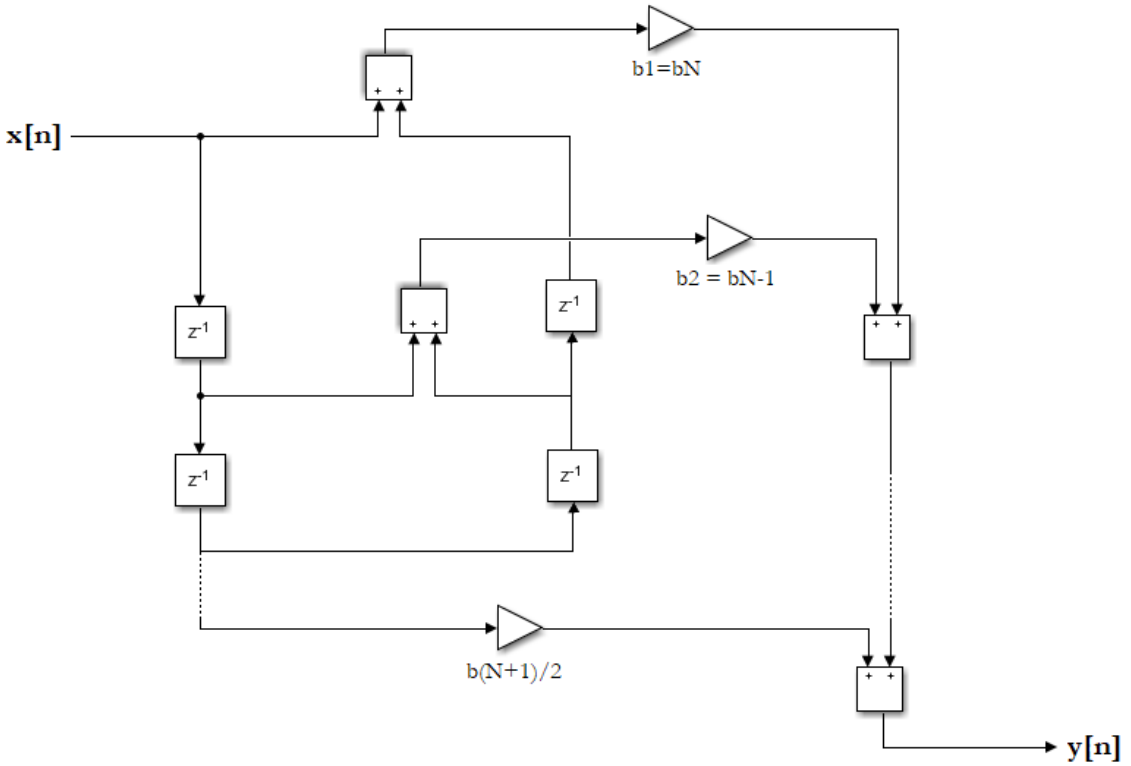

Figura 2.12: Estrutura de forma directa simétrica para um filtro FIR de comprimento N

Em suma, um filtro ideal é uma rede que permite que apenas sinais de certas frequências passem e bloqueie todas as outras. Dependendo da região de frequências permitidas ou não, os filtros são caracterizados como passa-baixo (*low-pass*), passa alto (*high-pass*), passa banda (*band-pass*), rejeita banda (*band-stop*) e passa tudo (*all-pass*).

Existem muitas necessidades de filtros eléctricos, alguns dos mais comuns sendo aqueles usados em aparelhos de rádio e televisão, que permitem sintonizar um determinado canal passando sua banda de frequências enquanto filtra os de outros canais. O filtro FIR é um filtro digital amplamente utilizado em aplicações de processamento de sinais digitais em vários campos, como imagem, instrumentação e comunicação.

O sinal do processador digital programável (*Programmable digital processor signal*, PDPS) pode ser usado na implementação do filtro FIR, mas hoje em dia, de forma a poder fazer a diferença no mercado, os novos sistemas de controle industrial precisam de demonstrar ter um elevado nível de desempenho, juntamente com flexibilidade e confiança. Ao mesmo tempo, o custo é uma questão fundamental [18].

Capítulo 3

Estado de Arte

Neste capítulo, é feita uma análise das soluções encontradas na literatura e do trabalho já existente de implementações de filtros feitas nos mais variados tipos de FPGA. São também abordados os conceitos discutidos já nos dois capítulos anteriores de forma a ter uma visão concisa e enquadramento do trabalho a ser desenvolvido.

3.1 Requerimentos

Devido aos requisitos de alto desempenho e à crescente complexidade do processamento de sinais digitais e aplicações multimédia, são necessários filtros com um comprimento elevado para aumentar o desempenho, de forma a produzir uma alta taxa de amostragem. Como resultado, as operações de filtragem são computacionalmente intensivas e mais complexas em termos dos requisitos de *hardware* [14].

Para reduzir o custo, o tempo de colocação no mercado deve ser reduzido, o preço de um dispositivo controlador deve ser barato e seu consumo de energia precisa ser reduzido. Essa redução de custos torna-se um desafio ainda maior visto que os sistemas de controlo industrial são baseados em algoritmos de controlo cada vez mais sofisticados, que requerem assim um elevado número de recursos de computação e tempo de execução cada vez menor. No entanto, ao desenhar um filtro de ordem elevada, muitos cálculos complexos são necessários, o que invariavelmente afecta o desempenho geral dos processadores de sinais digitais comuns termos de velocidade, flexibilidade de custos, estabilidade e assim por diante [18].

Para lidar com todos esses desafios, as FPGAs tornaram-se um meio extremamente económico de realizar e processar algoritmos de processamento de sinais digitais que sejam computacionalmente intensivos e melhorar o desempenho geral do sistema.

3.2 Implementação em ASIC *versus* FPGA

Ao desenhar um sistema, é importante decidir com antecedência qual circuito integrado será vai ser usado. Os circuitos ASIC e FPGA têm muitas características em comum, mas também muitas diferenças.

Filtros digitais podem ser então implementados em sistemas embutidos tais como micro controladores [19] e as FPGA[20]. Esta última é no geral conseguida através de uma *hardware description language* como VHDL or Verilog, ou implementação esquemática, existindo estudos propostos em literatura sobre esta implementação, mas requerem uma *licensed software package* [21] [22] [18] [23]. Esta tese não irá recorrer a tais sem ser o *software* nativo à placa Arduino MKR Vidor 4000 e ao Matlab.

3.2.1 Benefícios do uso baseado em um ASIC

Um dos aspectos mais importantes ao projectar um sistema é seu desempenho. O ASIC oferece efectivamente um desempenho a um nível cerca de 3,2 vezes maior que a FPGA, de acordo com um estudo comparativo feito a mais de 20 projectos diferentes [24].

Outro benefício do uso de um ASIC em vez da FPGA é o baixo consumo energético associado aos sistemas baseados em ASIC, visto que como explicado na secção 2.1, as FPGAs têm dentro delas muitos elementos lógicos para cobrir a implementação das mais variadas funções, o que por sua vez levam a um maior consumo de energia do que um projecto baseado em ASIC. Embora existam projectos de FPGA desenhados para consumidores que requeiram dispositivos com as vantagens da FPGA mas de baixo consumo energético, a maioria não o é [24].

Conceptualmente, a característica mais importante disponível num projecto baseado em ASIC e o que o torna mais preferível do que um projecto baseado em FPGA, é a sua flexibilidade. Apesar de os FPGAs modernos conterem os CLBs capazes de adoptar qualquer propósito, como foi listado na subsecção 2.1.6, estes blocos foram projectados para serem muito gerais, a fim de desempenhar muitas funções diferentes. Esta desvantagem de ter blocos para uso geral torna a FPGA menos desejada para muitos utilizadores, enquanto que por por outro lado, um ASIC pode ser projectado por um único utilizador para implementar qualquer circuito, bem como implementar blocos auto-especializados para a aplicação desejada.

3.2.2 Benefícios do uso baseado em uma FPGA

Além do ASIC, um FPGA também contém vantagens consideráveis. Um FPGA pode ser útil ao projectar um protótipo onde o processo de teste e avaliação pode ser executado em partes do projecto, mesmo quando o projecto não está totalmente concluído [24] e em contraste, as peças para a manufactura de um ASIC serão em princípio caras.

O baixo custo dos dispositivos FPGA torna-os disponível mesmo para amadores que nunca antes enverdaram pelo uso desta tecnologia. Por outro lado, o custo da ferramenta

para o ASIC é muito maior que o FPGA e portanto, menos acessível.

Uma FPGA fornece configurabilidade através da grande quantidade de lógica incorporada que pode ser usada para projectar qualquer circuito desejado. No entanto, a disponibilidade da lógica leva a que parte desta mesma não seja usada, o que, por sua vez, pode levar a menor desempenho e alto consumo de energia em comparação com um projecto em ASIC. O recurso de auto-configuração numa FPGA pode ajudar o *designer* a manter o seu *design* privado e não entregá-lo a terceiros, uma característica que ganha apelo com projectos que contêm informações confidenciais tais como códigos criptográficos [18].

Em suma com tudo o que já foi apresentado nos capítulos anteriores, a FPGA apresenta-se como a escolha mais poderosa devido a [18]:

- Tempo de colocação no mercado - A tecnologia FPGA oferece flexibilidade e recursos de prototipagem rápida, devido a maiores preocupações com o tempo de colocação no mercado. É possível testar uma ideia ou conceito e verificá-la em *hardware* sem passar pelo longo processo de fabricação do design ASIC personalizado.

  É fácil implementar mudanças incrementais e iterar num design de FPGA dentro de horas, em vez de semanas;

- Custo - A despesa de engenharia não recorrente (*nonrecurring engineering* - NRE) do *design* personalizado do ASIC excede em muito o custo das soluções de *hardware* baseadas em FPGA.

  Com a FPGA, isso significa que não existem custos de fabricação ou prazos de entrega longos para a montagem. Isso ocorre porque os requisitos do sistema geralmente mudam com o tempo, o custo de fazer alterações incrementais nos projectos de FPGAs é insignificante quando comparado à grande despesa de redesenhar um ASIC;

- Manutenção a longo prazo - Como discutido anteriormente na subsecção 2.1.6.2, as FPGA são actualizáveis em campo (*field programmable*) e não exigem tempo e despesas envolvidos na reformulação do ASIC. Os protocolos de comunicação digital, por exemplo, têm especificações que podem mudar com o tempo e as interfaces baseadas em ASIC podem causar desafios de manutenção e compatibilidade com a frente.

  Devido à natureza reconfigurável das FPGA, elas pode acompanhar as modificações futuras que possam eventualmente vir a ser necessárias. À medida que um produto ou sistema amadurece, é possível fazer aprimoramentos funcionais sem perder tempo a redesenhar o *hardware* ou a modificar o *layout* da placa;

- Desempenho - A natureza de paralelismo de *hardware* dos FPGAs excede o poder de computação dos processadores de sinal digital (DSPs), quebrando o paradigma da execução sequencial e realizando mais por ciclo de *clock*, dando assim um *trade-off* maior.

O controle de entradas e saídas (I/O) ao nível do *hardware* fornece tempos de resposta mais rápidos, como visto na subsecção 2.1.6.4, e funcionalidade especializada para atender aos requisitos do aplicativo. Isso novamente avalia e valida a natureza do paralelismo do FPGA;

- Confiabilidade - Devido ao facto de as ferramentas de *software* fornecerem o ambiente de programação, os circuitos das FPGA são realmente uma implementação "difícil" da execução do programa. Os sistemas baseados em processador geralmente envolvem várias camadas de conceitos para ajudar a agendar tarefas e compartilhar recursos entre vários processos.

  Para qualquer núcleo de processador, apenas uma instrução pode ser executada de cada vez, e os sistemas baseados em processador correm continuamente o risco de tarefas críticas de tempo se anteciparem, como explicado na subsecção 2.1.6.3.

  Como as FPGAs não usam sistemas operacionais, elas minimizam as preocupações de confiabilidade com verdadeira execução em paralela e o *hardware* determinístico dedicado a todas as tarefas.

3.3 Implementação em FPGA

O número de *taps*, ou comprimento, do filtro FIR (geralmente designadas como "N") é uma indicação de três variáveis:

- a quantidade de memória que vai ser requerida para implementar o filtro;
- o número de cálculos necessários;
- a quantidade de "filtragem"que este pode efectivamente fazer.

Visto que a complexidade de implementação cresce com a ordem do filtro e a precisão da computação, a realização de implementações em tempo real destes filtros com o nível desejado de precisão revela ser uma tarefa árdua. Mais *taps* significa mais atenuação da banda de parada, menos ondulações (*ripple)*, filtros mais estreitos etc [25].

Várias tentativas já foram, no entanto, postas em prática para implementação de filtros FIR em arquitecturas reconfiguráveis em tanto ASIC como FPGA.

Design sistólico (*inputs* e resultados moverem-se na mesma direcção a velocidades diferentes) representa um paradigma de arquitectura atraente para uma implementação *hardware* eficiente de aplicações DSP com exigências computacionais elevadas, sendo suportado pelos benefícios da sua simplicidade inerente, regularidade e modularidade da estrutura.

Em adição, possuem considerável potencial para retornarem um rácio elevado de transferência ao fazerem uso de processos concorrentes, *pipelining* ou ambos.

Para tirar partido deste processamento sistólico, vários algoritmos e arquitecturas foram sugeridos para a "sistolização" de filtros FIR, no entanto, os *multipliers* destas estruturas requerem uma porção maior da área do chip e consequentemente forçam a uma limitação do número de elementos de processamento que podem ser acomodados e a ordem máxima a que o filtro pode ser realizado.

Este constrangimento levou a um aumento da popularidade do uso de técnicas *distributed arithmetic* (DA) sem elementos *multipliers*, devido à alta taxa de transferência de processamento e a maior regularidade com que os resultados provaram ser *cost effective* e um *trade-off* positivo em termos de eficiência entre a área e o tempo das estruturas de computação.

Os principais operadores que são necessários para a computação com base em DA e produto interno, são uma sequência de acessos de *lookup tables* (LUT) seguidas de operações conjuntas de acumulação do *output* das LUT. Esta computação baseada em DA tem perfeita compatibilidade com as FPGA devido à sua estrutura lógica interna baseada em LUT como descrita na subsecção 2.1.2.

Em filtros FIR, uma das sequências de convolução deriva da sequência de amostras de inputs enquanto que a outra sequência deriva dos coeficientes da resposta de impulso do filtro.

Este comportamento do FIR faz com que seja possível usar técnicas com base em DA para implementação a partir da memória, resultante num output mais rápido pois armazena os resultados parciais pré computados em elementos de memória que podem ser lidos e adicionados para obter os resultados desejados.

O requerimento de memória para esta implementação dos filtros FIR no entanto, aumenta conforme a ordem do deste. [26]

3.4 Implementação pelo método *Kaiser window*

No estudo realizado em [16], foi investigado uma implementação de filtros FIR digitais com esquemáticos das três diferentes arquitecturas fundamentais estudadas na subsecção 2.4.4. As implementações dos esquemáticos foram realizadas numa placa FPGA de modelo Altera DE2-115, sem exigir quaisquer outros tipos de *software* complementares excluindo o Matlab, cujos resultados nas tabelas 3.1, 3.2 e 3.3 demonstram e comprovam uma aplicação bem sucedida.

Os filtros foram primeiro projectados em Matlab, através da kaiser window, e depois testados e aplicados à placa, algo que depois de uma familiarização com a tecnologia é feito no capítulo seguinte 4.

Estruturas	N = 11	N = 21	N = 31	N = 41	N = 51
Directa	150	281	360	439	521
Transposta	80	168	248	328	411
Simétrica	124	242	367	492	608

Tabela 3.1: Comparação das estruturas em termos de lógica total usada para vários comprimentos [16]

Estruturas	N = 11	N = 21	N = 31	N = 41	N = 51
Directa	80	120	120	120	120
Transposta	80	160	240	320	400
Simétrica	80	160	240	320	400

Tabela 3.2: Comparação das estruturas em termos do número total de registos usados para vários comprimentos [16]

Estruturas	N = 11	N = 21	N = 31	N = 41	N = 51
Directa	22528	32848	33008	33168	33328
Transposta	22528	43008	63488	83968	104448
Simétrica	12288	22528	32768	43008	53248

Tabela 3.3: Comparação das estruturas em termos de *bits* de memória usada para vários comprimentos [16]

Foram assim implementadas três estruturas FIR fundamentais para vários comprimentos de filtro, desde N = 11 a N = 51, para serem de seguida comparadas em termos de elementos lógicos totais, número de registos e memória ocupada, sendo compiladas no Quartus-II.

Foi concluído que a menor quantidade de elementos e registos lógicos totais são obtidos para estruturas de forma directa e de forma directa transpostas, respectivamente. Quanto ao menor número total de bits de memória, os melhores resultados são obtidos para a estrutura de forma directa simétrica para comprimentos de filtro mais baixos, e para a estrutura directa no caso de comprimentos de filtro mais altos.

3.5 Implementação pelo método *Equiripple*

O filtro *Equiripple*, como o nome sugere, possui ondulações iguais na banda de passagem e na de *stop*, o que significa que a distorção do sinal que ocorre no limite da banda de passagem devido à presença de uma grande ondulação (*ripple*) é evitada com este tipo de *design*, mas este possui uma grande faixa de transição, o que limita assim a largura total da banda de passagem.

No artigo [18], foi escolhido o método *Equiripple* porque:

- cumpre as especificações com o menor número de coeficientes;
- usa menos quantidade de recursos na FPGA para implementação;

- o erro de aproximação ponderada entre a resposta de frequência desejada e a real é espalhado uniforme-mente entre a banda de passagem e a banda de paragem do filtro, minimizando assim o erro;
- os desvios das bandas podem ser especificados separadamente.

Esta abordagem de *design* de filtro usando o método *Equiripple* aprimora selecções fáceis para especificar métodos de banda de passagem, ordem de filtro e *design*, além de fornecer uma *filter response* de especificação individual. A ferramenta fdatool do Mathworks, como mostrado para o filtro passa-baixo na figura 3.1, é usada para gerar tabelas de coeficientes de modelo MATLAB com base nas especificações.

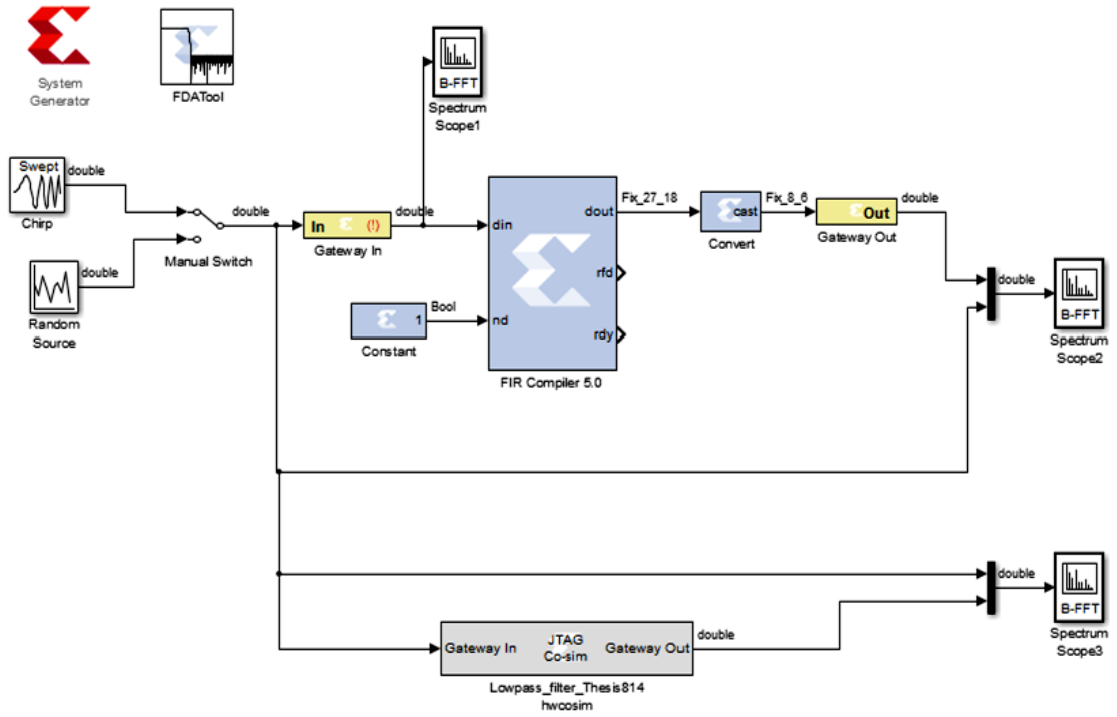

Figura 3.1: Filtro FIR passa baixo [18]

Depois de projectar os filtros com base nas especificações da Matlab, o pacote de *software* Xilinx fornecido pela placa Spartan-6 FPGA, o System Generator é usado para a implementação apropriada do filtro FIR FPGA para passa-baixo, conforme mostrado nas figuras 3.2, 3.3 e 3.4.

Os resultados das simulação mostram que as formas de onda de saída obtidas nas simulações de *software* correspondem àquelas da implementação do FPGA, respectivamente, usando co-simulações de *hardware* [18].

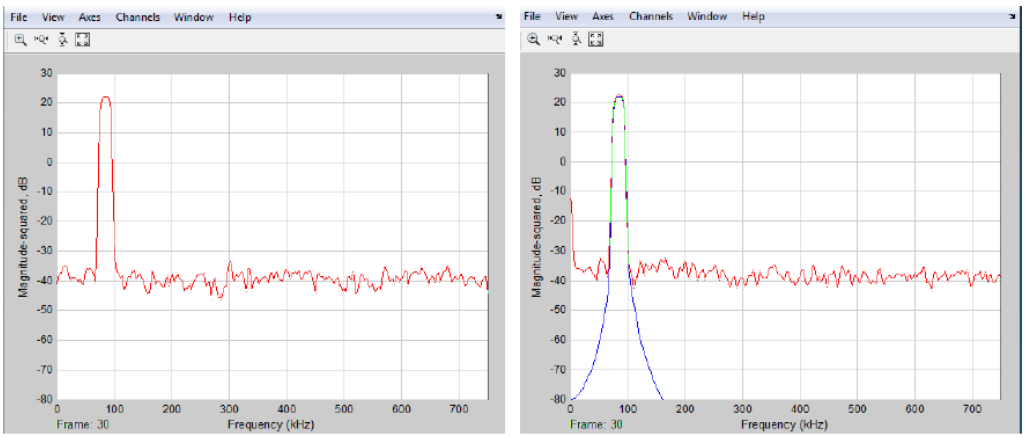

Figura 3.2: *Input* abaixo de f_C usando *chip source* e o *Output* do passa baixo usando o Matlab [18]

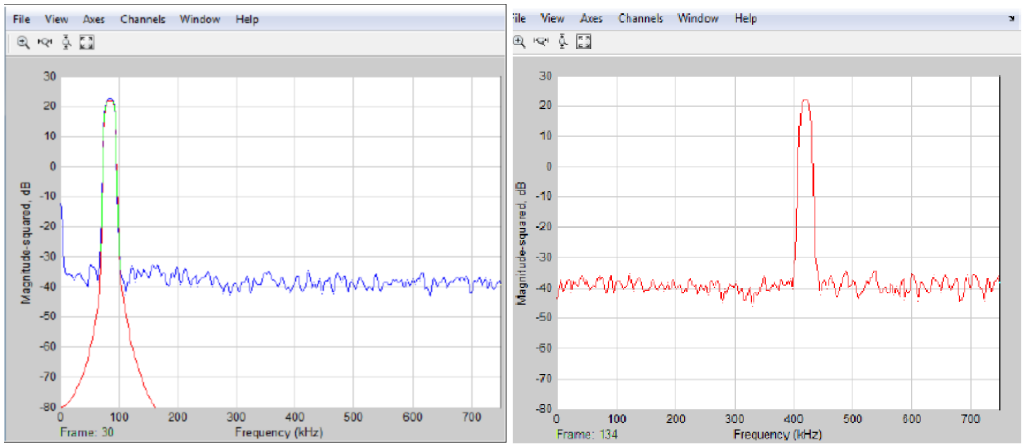

Figura 3.3: *Output* do passa baixo usando o Matlab e o *Input* acima de f_C usando *chip source* [18]

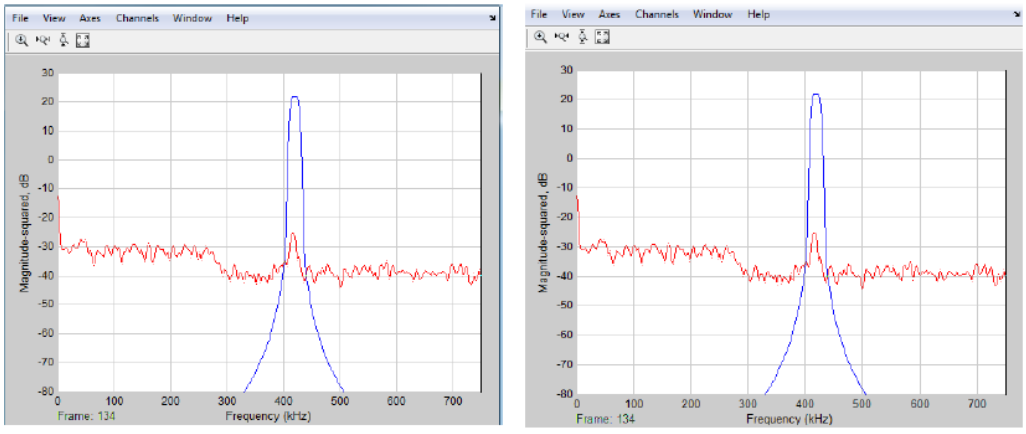

Figura 3.4: *Output* do passa baixo usando o Matlab e FPGA respectivamente [18]

Esta aproximação pelo método *Equiripple* embora desejável não foi implementada como demonstrada em literatura devido devido à falta do *software* necessário, pois este é inerente à FPGA Spartan-6 [18] entre outras.

Com respeito aos constrangimentos ditados pela tecnologia FPGA seleccionada, foi escolhida a implementação com recurso ao método *Kaiser window*, o que tirou descartou a necessidade de *software* adicional e externo para a realização do projecto.

Capítulo 4

Implementação de um filtro configurável

Para alcançar os objectivos da presente dissertação, dividiu-se o desenvolvimento das soluções em várias etapas. Este capítulo focar-se-á em cada uma dessas etapas: uma primeira inicial de projecção do filtro a implementar, as primeiras impressões com a placa, a problemática encontrada com o ambiente escolhido, as soluções adoptadas e as primeiras tentativas de programação na FPGA.

Por fim, a elaboração de um esquemático e o *set up* da *breadboard* com a FPGA de forma a poder correr e serem visualizados os esquemáticos de filtro.

4.1 *High Level Analysis*

O *software* MATLAB produzido pela *MathWorks Company* é um dos mais úteis e poderosos *software environments* à disposição de cientistas e engenheiros [27]. Neste trabalho, foi a ferramenta usada para projectar e simular um filtro FIR digital. Estes filtros foram desenhados e avaliados utilizando no Matlab uma *interface* denominada fdatool, *filter design and analysis tool*.

Nesta secção, apresenta-se as simulações realizadas no ambiente Matlab para mais à frente comparar com a realização destes mesmos na placa FPGA.

4.1.1 Filtro FIR Passa Baixo

Para poder simular um filtro digital e implementá-lo numa FPGA, primeiro foi preciso projectar esse mesmo filtro usando o *software* já referenciado, tal como em [16]. Depois de uma breve familiarização com a fdatool, foi então desenhado um filtro passa baixo FIR, de ordem 4 baseado na *Kaiser window* [16], com as seguintes especificações, demonstradas na figura 4.1:

- beta = 1;
- comprimento do filtro N = 5 (ou ordem de filtro = 4);
- frequência de amostragem f_S=1000 Hz;
- frequência de corte f_C = 100 Hz.

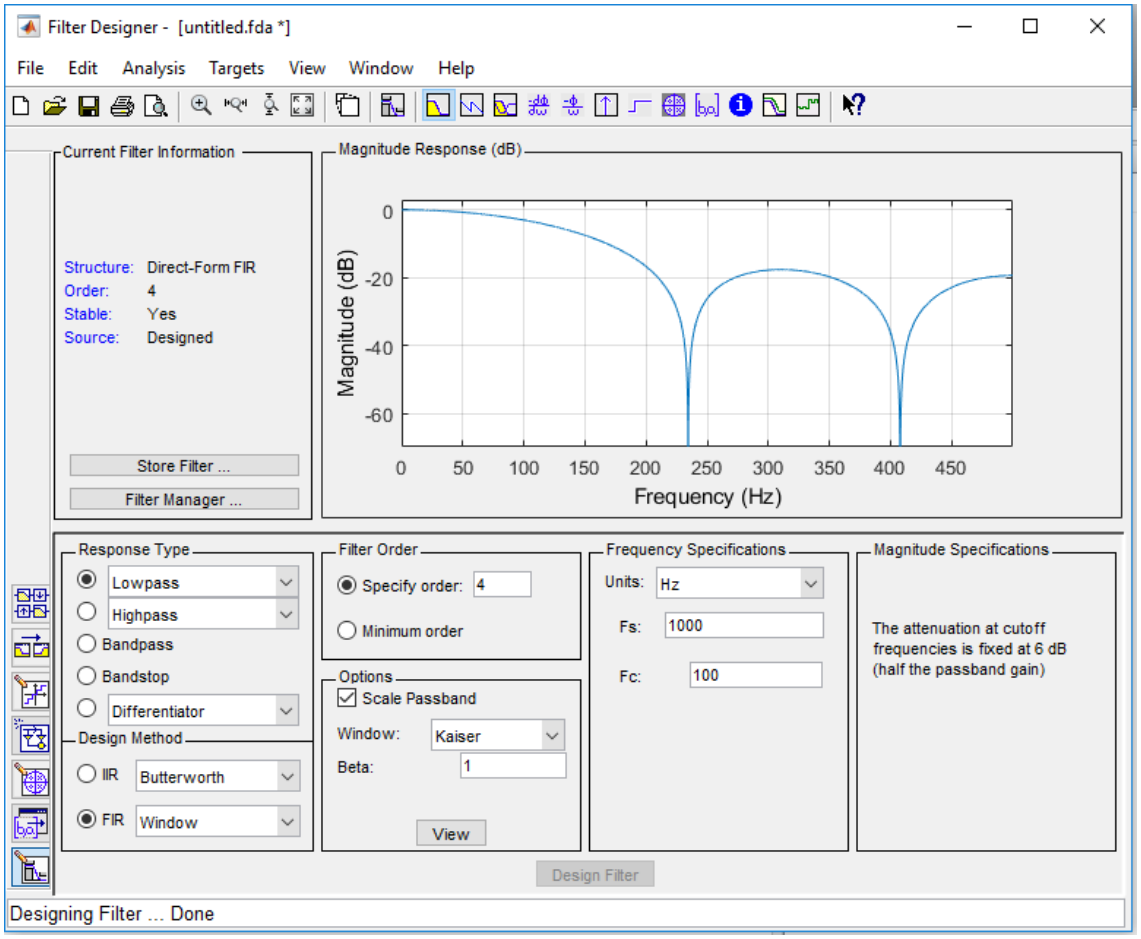

Figura 4.1: Filtro FIR Passa Baixo projectado com a *Kaiser window*, N = 5 (ou ordem 4) e frequência de corte fc=100 Hz

Os cinco coeficientes do filtro não inteiros encontrados através da fdatool são apresentados na figura 4.2 tendo sido depois então arredondados para serem inteiros, na imagem 4.3, sendo eles: 151, 223, 252, 223 e 151.

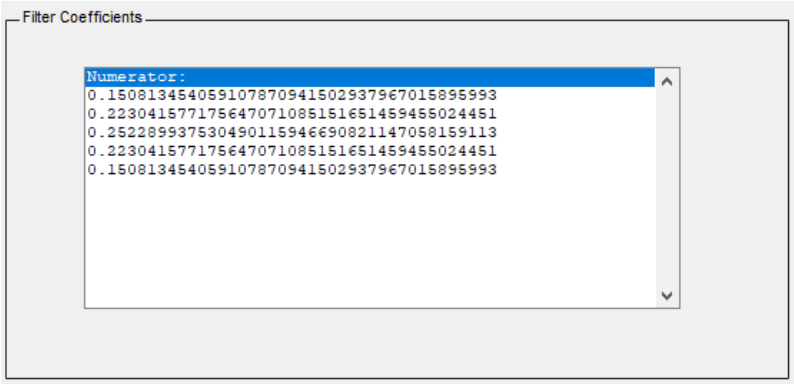

Figura 4.2: Coeficientes não-inteiros do filtro

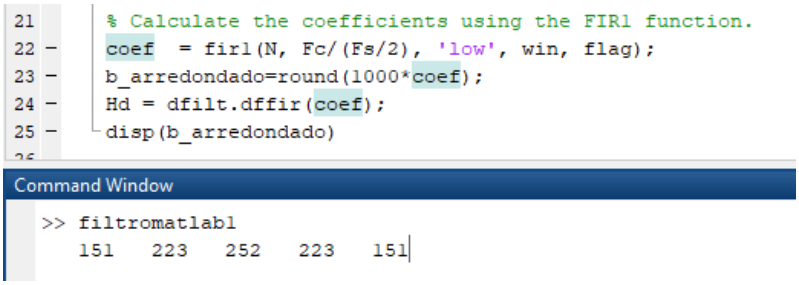

Figura 4.3: Coeficientes de filtro arredondados a inteiros para o filtro desejado

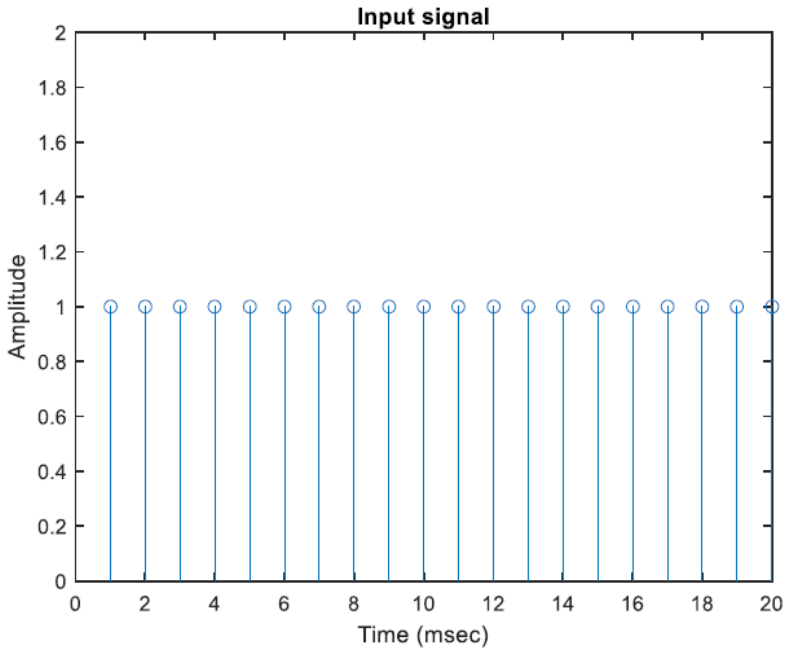

Figura 4.4: Sinal de entrada

O sinal de teste que foi usado como entrada no filtro projectado para o exemplo de simulação e realização é mostrado na figura 4.4.

Quando o sinal de entrada da figura 4.4 é aplicado ao filtro projectado, que é definido pela sua resposta ao impulso visto na imagem 4.5, o sinal de saída é obtido como demonstrado na figura 4.6.

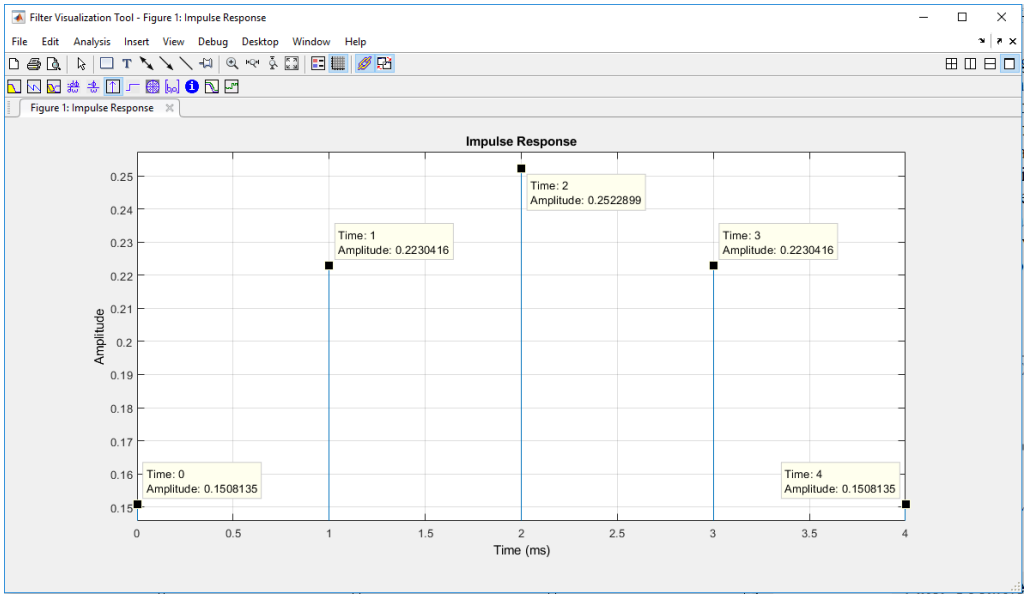

Figura 4.5: Resposta ao impulso do filtro desenhado (sem arredondamento)

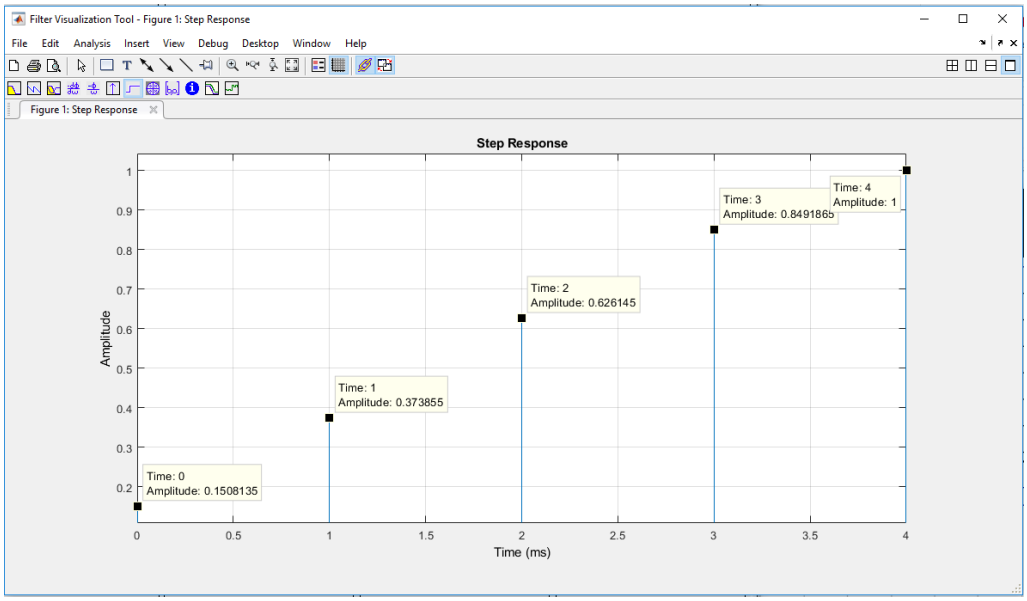

Figura 4.6: Resposta ao degrau do filtro desenhado (sem arredondamento)

É possível ver o resultado final obtido e observar que o valor filtrado do *steady state* é 1000 (em decimal), o qual será usado para avaliar a implementação proposta.

A secção seguinte foca-se então na realização de um filtro de forma a que consiga apresentar um comportamento semelhante este, mas no *hardware* da Arduino MKR Vidor 4000.

4.2 Manuseamento da Vidor

O desafio inerente à placa Arduino MKR Vidor 4000 foi a junção das duas plataformas distintas com a junção da FPGA ao *hardware* Arduino, o que deveria permitir um processamento de áudio e vídeo que excedesse o de um micro controlador.

A placa Arduino Vidor 4000 consiste de:

- um micro controlador ATSAMD21G18A, com 256KB de *flash*, 32KB de SRAM e clock de 48Mhz;
- uma FPGA Cyclone 10LP (10CL016) com 15408 elementos de lógica , 504KB de RAM e 56 multiplicadores de 18 × 18;
- uma FLASH SPI de 16MB;
- uma SDRAM de 64MB (4M x 16 bits);
- um módulo Qifi/BLE NINA W102 integrado num micro controlador ESP32 de *dual core*;
- um chip criptográfico ATECC508A com o intuito de melhorar a velocidade de processamento em ligações seguras;
- conectores MiniPCIe, USB, bateria, I2C, MKR, MIPI para a câmara e HDMI para output de vídeo.

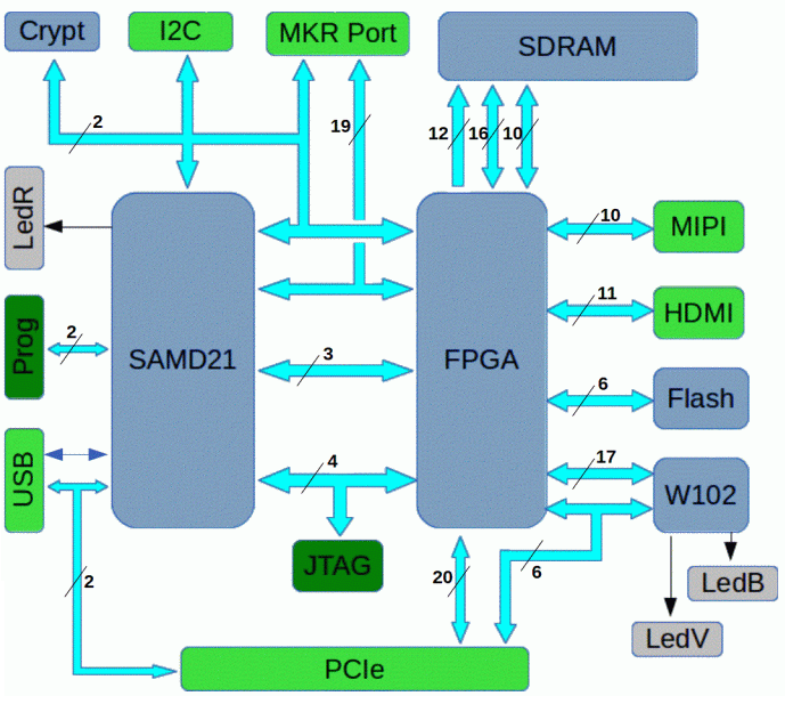

Figura 4.7: Estrutura interna MKR Vidor 4000 [10]

Esta estrutura, representada na figura 4.7, foi desenhada pela equipa com o propósito de colocar a placa FPGA no centro do sistema, de forma a ter acesso a todos os recursos, permitindo ao utilizador a liberdade de os gerir ou os alocar para para o processador.

Visto que o micro controlador e o módulo *Wifi* têm a sua própria memória não volátil para executar o programa, a configuração inicial da FPGA é então carregada com cada iniciação da memória FLASH. Ao ligar, temos então o seguinte:

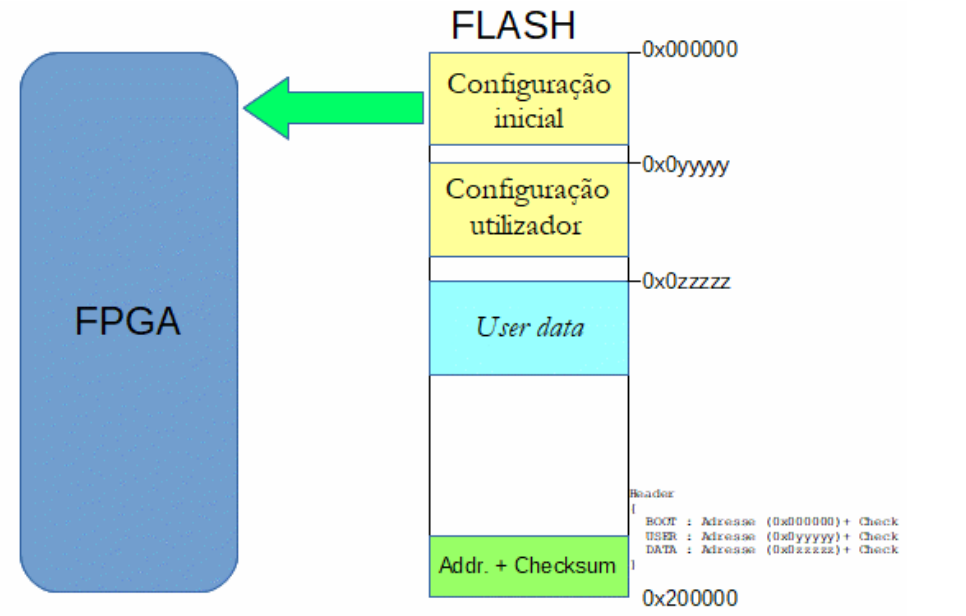

Figura 4.8: Configuração inicial da FPGA [10]

De momento, os *source files* desta configuração não foram publicados pela Arduino, portanto não lhes foi possível aceder. No entanto, pode-se dizer que permite:

- exibir logótipo "ARDUINO ™" na saída HDMI;
- ler e escrever no *flash* para actualizar as 2 configurações possíveis, os de *factory settings* e de utilizador, tal como dados inseridos;
- alterar para a configuração de utilizador.

Portanto, é através dessa configuração carregada na inicialização que poderemos actualizar vários tipos de configurações do utilizador. A Arduino disponibilizou no momento 2 configurações, *VidorGraphics* e *VidorPeripherals*. A primeira possibilita:

- desenhar na saída HDMI;
- fazer *display* do que uma câmera ligada ao MIPI vê e reconhecer códigos QR.

E a segunda biblioteca dá acesso a um maior número de interfaces via a porta MKR (links de série UART / SPI / I2C, codificadores em quadratura, etc).

4.2.1 Problemática

Infelizmente, no decorrer do trabalho desenvolvido nesta tese e do processo de familiarização com a placa em questão, entre as quais contacto directo com variados engenheiros (tanto da Intel e *hobbyists*) e membros da equipa Arduino, tornou-se claro que a MKR Vidor 4000 é uma placa pioneira e com tempo de vida ainda muito curto (menos de 6 meses aquando do início desta tese), contendo pouco ou virtualmente ainda nenhum suporte para os utilizadores.

Devido ao constrangimento do tempo de escrita e entrega desta dissertação, foi tomada a decisão de seguir em frente, apesar das promessas de apoio por parte da equipa Arduino de maior suporte, tanto a nível de interactividade da Arduino IDE e de um Visual Editor que, aquando da escrita desta secção (Agosto de 2019), não tinha ainda sido apresentado ao público nem demonstrava sinais de o ser tão cedo.

Sendo assim que, oficialmente, a continuidade de apoio por parte da equipa Arduino para esta placa foi *put on hold* ou adiada, visto que se apresentou como um desafio maior do que haviam considerado e continuam a explorar soluções mais simples, de acordo com informação divulgada.

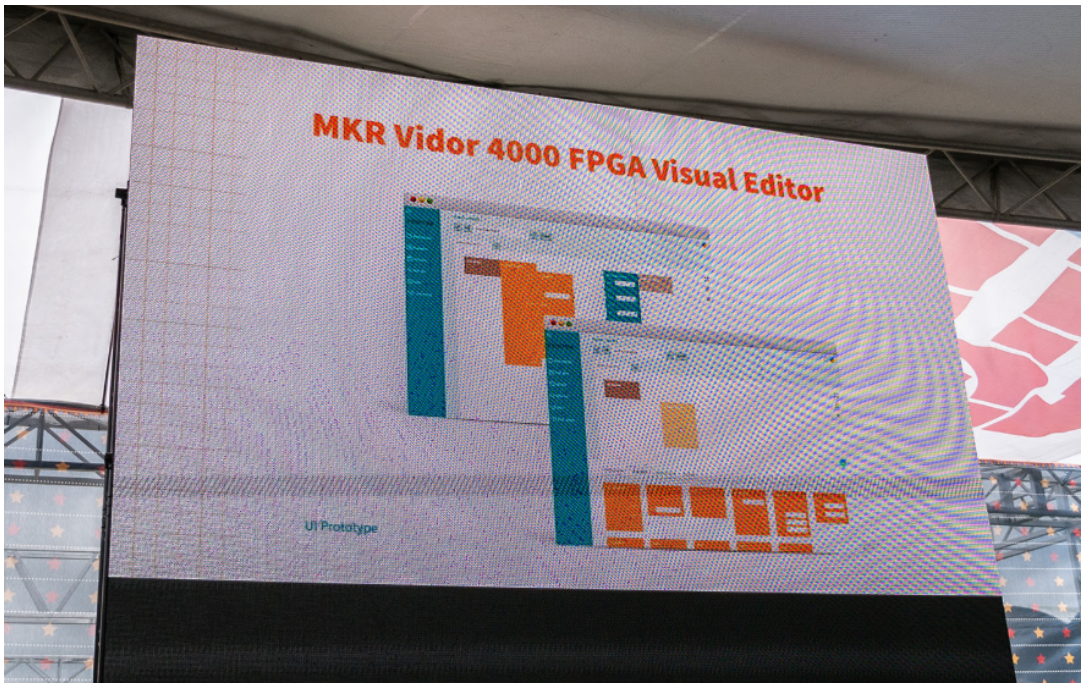

Figura 4.9: Protótipo apresentado "MKR VIdor 4000 FPGA Visual Editor" [28]

Acrescentado a esta falta de suporte, foi deixado explícito que ela ***não*** foi feita para que o utilizador programe directamente a FPGA através do Arduino IDE, tendo sido então forçosamente tomado um percurso alternativo para poder prosseguir com o trabalho e a implementação. Num futuro próximo, talvez haverá a possibilidade de definir a conexão entre diferentes blocos lógicos por parte da Arduino. Assim, não foi possível programar directamente o FPGA através da Arduino IDE.

Na próxima secção 4.3 veremos a alternativa tomada para programar na FPGA, mas apesar deste percalço, foram feitos testes básicos à Vidor a um nível exclusivamente de Arduino.

Seguindo o guia disponibilizado na página do material [29], foi feito o *download* do IDE e seguidamente, procurando por SAMD, e a instalação da placa correspondente no ambiente de desenvolvimento ("**Tools / Type of card ..**" e seleccionando *Arduino MKR Vidor 4000*), fazendo por fim a ligação através de uma configuração da porta COM correspondente.

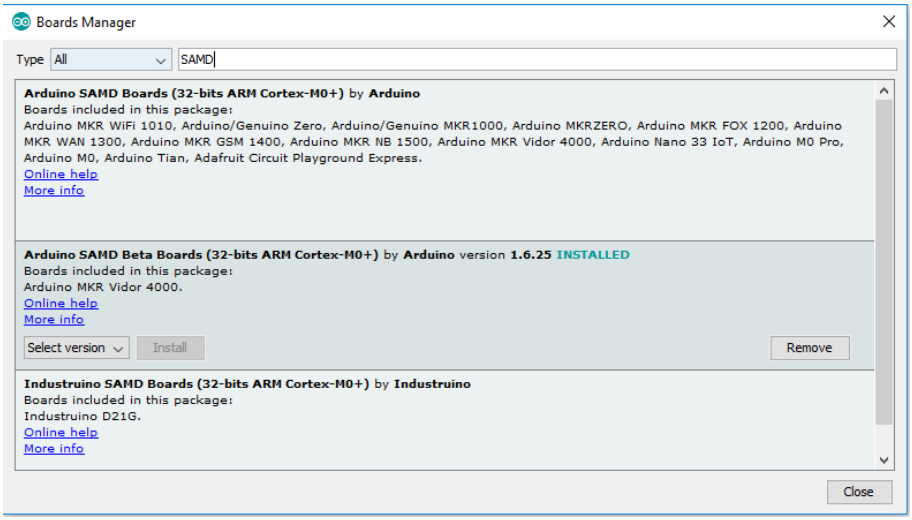

Figura 4.10: Escolha do *environment* correcto

Fazendo de seguida a compilação e o *upload* de um *sketch* básico e conhecido da biblioteca Arduino, o "blink" (**File → Examples → 01. Basics → Blink**), verificamos que tudo funciona correctamente. Uma simples alteração na função **delay()** de 1000 (1 segundo) para 200 resulta num piscar mais rápido do (único) LED embutido na placa. É assim feito o *download* do programa para a memória do micro controlador SAMD21, sem alterar a configuração da FPGA vista na imagem 4.11.

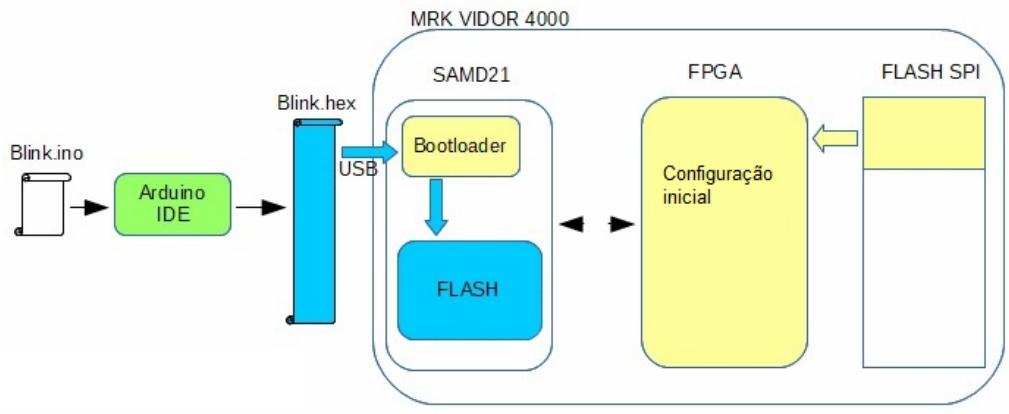

Figura 4.11: *Download* do Blink.ino [10]

De seguida as bibliotecas correspondentes à placa, encontradas com a *tag* Vidor, foram instaladas como se pode provar na imagem 4.12, e foi possível verificá-lo observando a saída da porta COM correspondente.

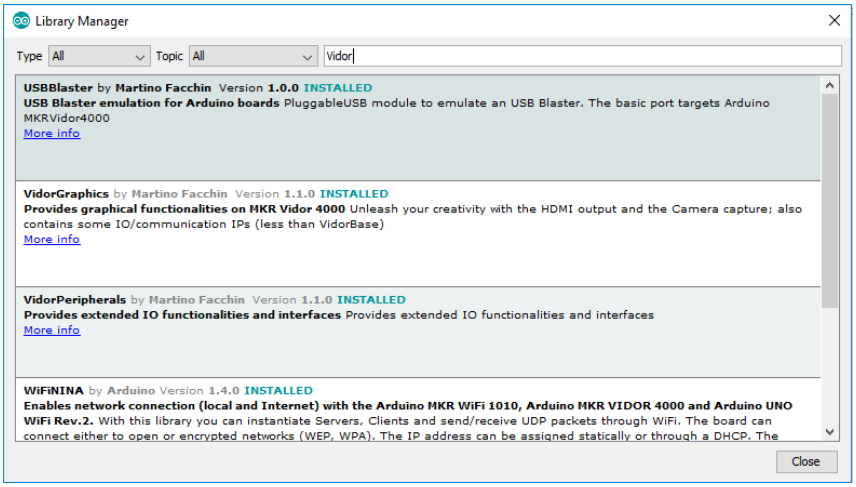

Figura 4.12: *Upload* das bibliotecas

Foi agora não só programado o SAMD21 FLASH, mas também reconfigurada a FPGA com a configuração do VidorPeripherals. Existe entre o micro controlador e a FPGA uma ligação do tipo JTAG (Joint Test Action Group) um *standard* da indústria originalmente destinado ao teste de placas electrónicas e, em seguida, adaptado à programação de componentes tais como as FPGA. Nesta configuração ela é usada para estabelecer comunicação entre o *bootloader* com um bloco lógico implementado na configuração inicial do FPGA. Este pode depois solicitar-lhe a leitura e gravação de dados na memória FLASH SPI.

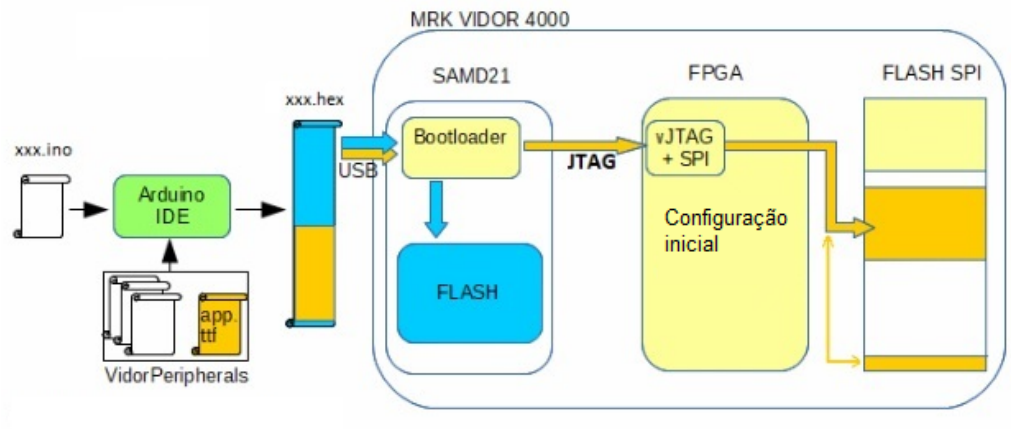

Figura 4.13: Iniciar a FPGA [10]

Ao fazer a compilação do *sketch*, o compilador incluirá no ficheiro gerado a configuração (designada app.ttf) a ser posteriormente gravada na FPGA. Tendo a configuração posicionada num endereço específico, o *bootloader* poderá transferi-la para a FPGA e escrever o FLASH SPI.

Ao ligar a MKR Vidor, a configuração inicial da FPGA é carregada e o micro controlador irá executar o código da listagem 4.1.

Listagem 4.1: Primeiras Instruções

```
1 // Let's start by initializing the FPGA
2     if (! FPGA.begin ()) {
3     Serial.println ("Initialization failed!");
4     while (1) {}
5 }
```

A função *FPGA.begin ()* vai em primeiro lugar fornecer à FPGA um sinal de *clock* para que ele possa iniciar a **enableFpgaClock ()**, que em seguida inicia a ligação JTAG para comunicação com a FPGA através de **jtagInit ()** .

Por fim, enviará à FPGA pela porta JTAG o comando 0x00000003, solicitando que carregue a configuração que foi enviada pelo utilizador, ou seja o ficheiro app.ttf escrito anteriormente no FLASH.

Todas as outras instruções para a FPGA usando as bibliotecas instaladas e *sketches* do *backlog* Arduino, quer seja pedir para desenhar um círculo ou configure uma porta COM, serão enviadas e a resposta recebida através dessa mesma ligação JTAG [10].

4.3 Quartus *Software*

Sendo a FPGA da placa um ALTERA Cyclone 10LP, programação directa obrigou ao uso do *software* Quartus, a ferramenta correspondente para desenvolvimento de projectos de FPGAs ALTERA.

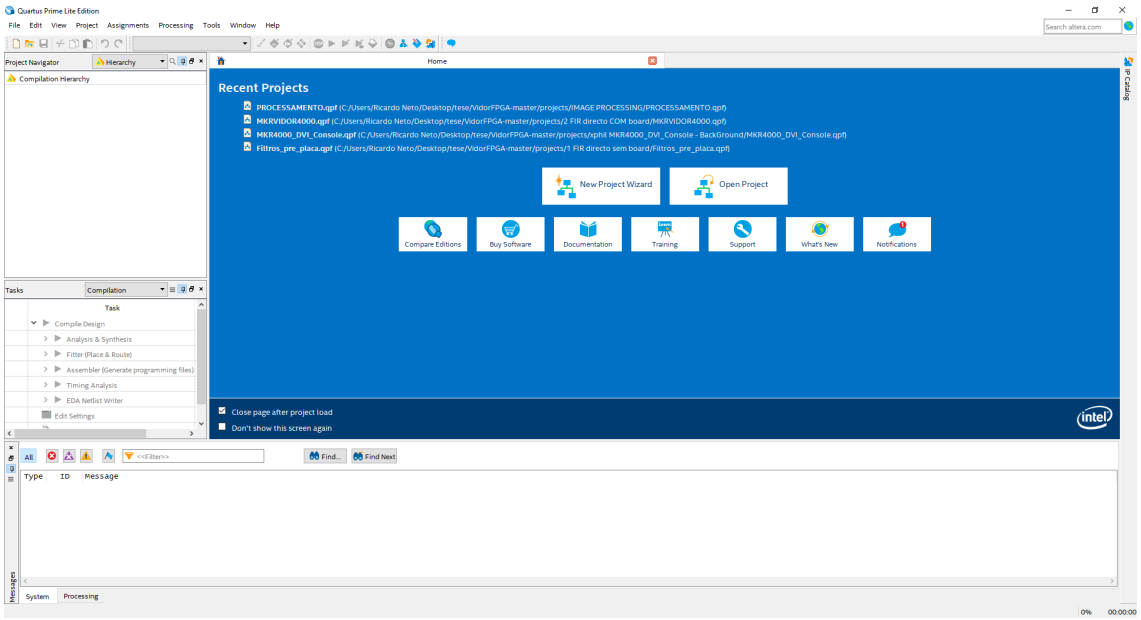

Figura 4.14: Quartus Prime 18.1 Lite Edition

Esta ferramenta, produzida pela Altera, é um *software* de *design* de dispositivos lógicos programáveis, e que permite a análise e síntese de projectos de HDL [30]. É então agora necessário o uso de linguagens HDL, neste caso o Verilog, que foi mantido pela Arduino para a programação desta FPGA. Existindo diversas convenções para Verilog, foram seguidos os exemplos dados no código disponibilizado pela equipa para manter homogeneidade.

Neste estudo, o Quartus Prime é usado para implementar esquematicamente os filtros digitais, compilar e simular os filtros digitais e fazer upload dos arquivos necessários para a placa FPGA. Foi então feito o *download*, criada uma conta e utilizada a edição mais actualizada, Quartus Prime 18.1 Lite Edition de 64 bits, como demonstrado na imagem anterior 4.14.

Para dar início a um projecto básico na FPGA, é necessário haver um projecto base com todas as configurações I / O da Arduino MKR Vidor, que foi disponibilizado pela equipa em [31]. É importante usar esta configuração prévia de forma a saber quais são as ligações internas da FPGA já declaradas pela equipa de forma a não haver uma má configuração que ponha em risco o material.

Os componentes da placa **são sensíveis** e basta um valor elevado ou errado numa tensão nos pinos para causar estragos, existindo também o risco de causar um curto-circuito devido a uma configuração incorrecta carregada para a FPGA.

O módulo de maior nível, a entidade de topo, já definida para este projecto é o **MKRVIDOR4000_top**. Este ficheiro, escrito em Verilog e no topo da hierarquia como visto na figura 4.15 define:

- o nome de cada sinal I / O e a sua direcção, seja *input*, *ouput* ou *inout*;
- as ligações internas ou *wires*, *buffers*, registos, etc;
- instâncias de outros módulos a declarar;
- regras combinatórias (*assign*) ou sequenciais (*always*, *begin*, *end*) .

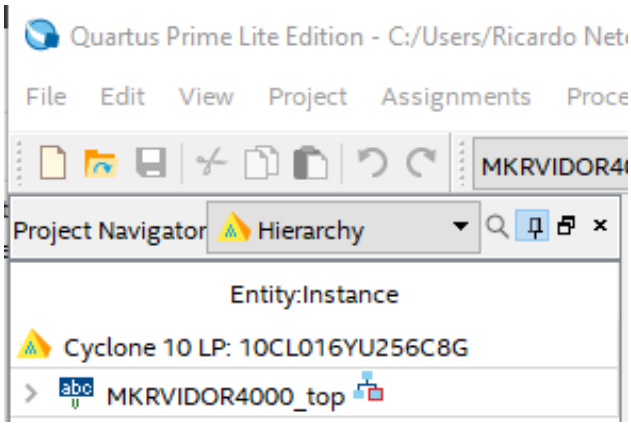

Figura 4.15: Ficheiro de topo do projecto com ligações já definidas

Além de uma declaração inicial de todos os *wires* e respectivas direcções, os sinais do sistema e ligações entre os diferentes componentes da Vidor, o ficheiro de topo **MKRVIDOR4000_top.v** inclui ainda as declarações em Verilog de um PLL e o oscilador interno da FPGA, cujo código pode ser visto na listagem 4.2.

Listagem 4.2: Código para oscilador interno e para PLL

```
// internal oscillator
cyclone10lp_oscillator    osc
  (
  .clkout(wOSC_CLK),
  .oscena(1'b1));

// system PLL
SYSTEM_PLL PLL_inst(
  .areset(1'b0),
  .inclk0(wCLK8),
  .c0(wCLK24),
  .c1(wCLK120),
  .c2(wMEM_CLK),
   .c3(oSDRAM_CLK),
  .c4(wFLASH_CLK),

  .locked());
```

Outro ficheiro indispensável é aquele que vincula os sinais de I / O com todos os *pins* da FPGA, definindo também os níveis de tensão desses sinais, sendo imperativo que este não seja modificado. É possível visualizá-lo ao selecionar "*Assignments* → *Assignments Editor*".

	tatu	From	To	Assignment Name	Value	Enabled	Entity	Comment	Tag
1	✔		iCLK	Location	PIN_E2	Yes			
2	✔		iRESETn	Location	PIN_E1	Yes			
3	✔		oSDR...[11]	Location	PIN_E10	Yes			
4	✔		oSDR...[10]	Location	PIN_B13	Yes			
5	✔		oSDR...R[9]	Location	PIN_C9	Yes			
6	✔		oSDR...R[8]	Location	PIN_E11	Yes			
7	✔		oSDR...R[7]	Location	PIN_D12	Yes			
8	✔		oSDR...R[6]	Location	PIN_D11	Yes			
9	✔		oSDR...R[5]	Location	PIN_C14	Yes			
10	✔		oSDR...R[4]	Location	PIN_D14	Yes			
11	✔		oSDR...R[3]	Location	PIN_A14	Yes			
12	✔		oSDR...R[2]	Location	PIN_A15	Yes			
13	✔		oSDR...R[1]	Location	PIN_B12	Yes			
14	✔		oSDR...R[0]	Location	PIN_A12	Yes			
15	✔		oSDR...A[1]	Location	PIN_B10	Yes			
16	✔		oSDR...A[0]	Location	PIN_A10	Yes			
17	✔		oSDR...CASn	Location	PIN_B7	Yes			
18	✔		oSDR..._CKE	Location	PIN_E9	Yes			
19	✔		oSDR..._CSn	Location	PIN_A11	Yes			
20	✔		bSDR...[15]	Location	PIN_B6	Yes			

Figura 4.16: Ficheiro das ligações de todos os sinais I / O (lista incompleta devido à sua extensão)

Após uma compilação do projecto é possível ver a arquitectura gerada com ao seleccionar "*Tools* → *Netlist viewers* → *RTL viewer*", verficicando-se o código da listagem 4.2. A imagem 4.17 contém blocos extra (TUTO_Schematic) do projecto implementado na secção seguinte 4.4.

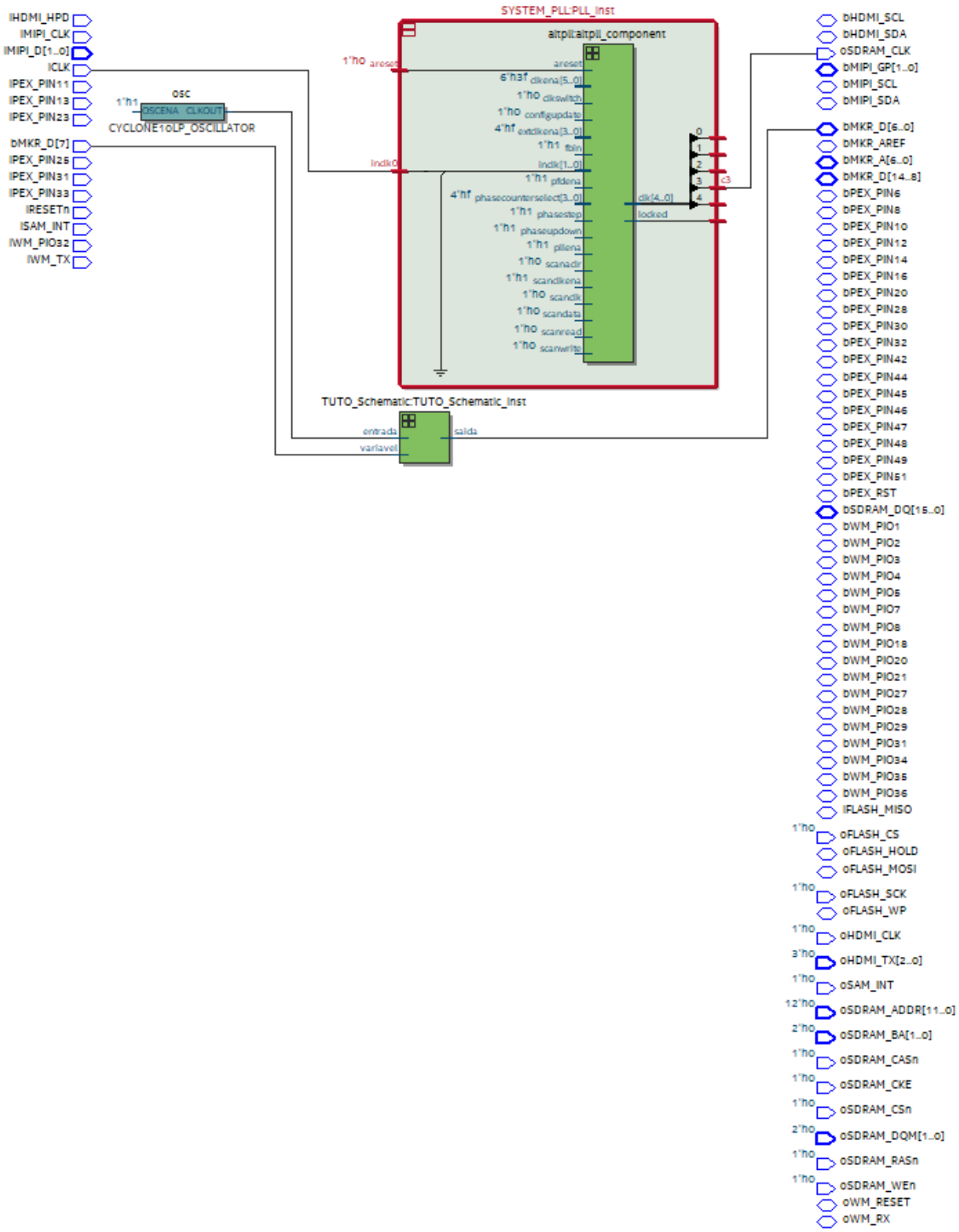

Figura 4.17: O *clock input* 48Mhz iCLK originando do SAMD21 atravessa o PLL gerando um *clock* de 100Mhz ligado à SDRAM

4.4 *Blink* com um interruptor

Como teste inicial, foi implementado o programa "blink" na FPGA usando um bloco lógico AND e um Counter, e tendo como *inputs* o *clock* interno da placa e um fio ligado ao VCC. Foi adicionado um esquemático a este projecto, acedendo em "*Make File → New*" e seleccionando "*Block Diagram / Schematic File*", abrindo assim uma nova *sheet* e criando um esquemático novo.

O seu funcionamento é o seguinte: o *clock* do oscilador interno da FPGA, que é usado como *input*, tem uma frequência bastante alta de cerca de 80MHz, foi então usado este LPM_COUNTER em vez da PLL como divisor de frequência.

A cada ciclo do *clock*, o contador será incrementado só se o *input* denominado *ival*, que externamente será controlado por um interruptor, se encontrar a 1 (devido ao AND). Ou seja, o LED vai ser aceso à frequência de saída do contador só se o interruptor se encontrar ligado. Criando todas as ligações e blocos necessários, ficamos com o esquemático da imagem 4.18, que depois de ser gravado é incluído na entidade de topo do projecto, **MKRVIDOR4000_top.v**.

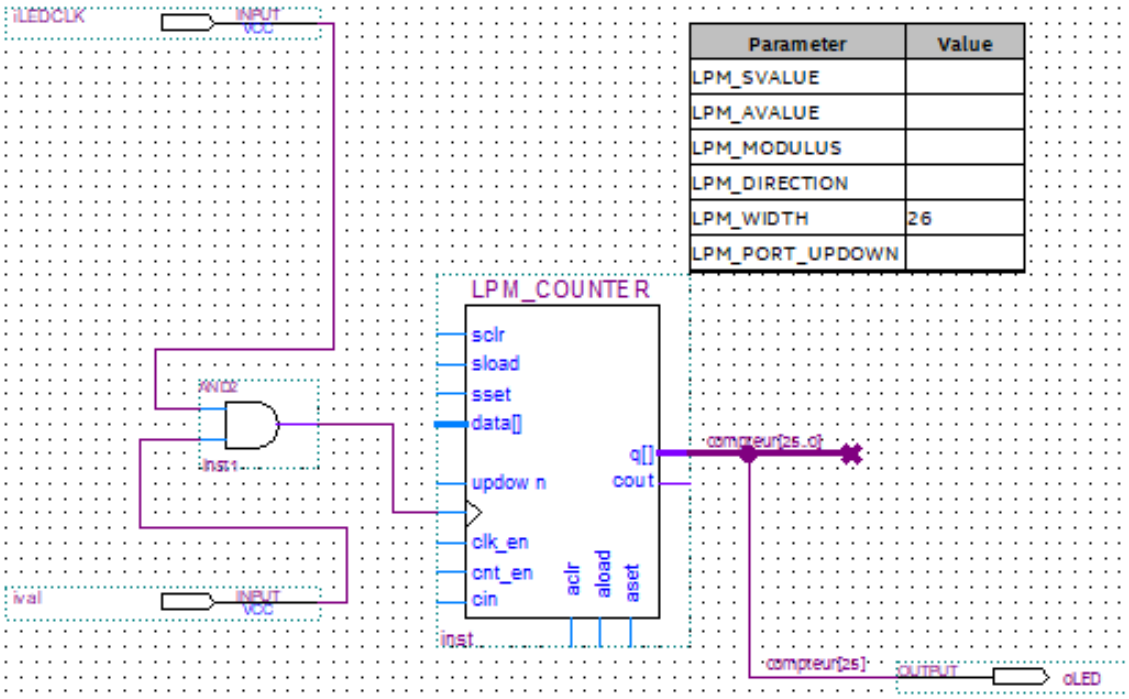

Figura 4.18: *Blink* com contador e porta AND

Assim, cada um dos bits do contador irá alterar a sua frequência, e para piscar uma vez a cada segundo será necessário n = 25, daí o *bus* escolhido para a saída.

$$Frequencia_{numerodebits} = \frac{Frequencia_{input}}{2^{(n+1)}} \tag{4.1}$$

Adicionando o esquemático criado anteriormente como instância através de uma declaração em Verilog, como mostrado na listagem 4.3, imediatamente a seguir ao código da listagem anterior 4.2, o projecto pode ser compilado e estará pronto para ser enviado para a placa.

Listagem 4.3: Código a implementar para incluir o novo esquemático

```
1 TUTO_Schematic TUTO_Schematic_inst
2 (
3 .entrada (wOSC_CLK),
4 // The clock input is connected to the internal oscillator 80Mhz
5 .saida (bMKR_D [6]),
6 // The LED output is connected to pin 6 of the MKR port
7 .variavel(bMKR_D [7])
8 //input ligado ao pin 7, valida o clock através da AND
9 );
```

Esta compilação gera um ficheiro denominado **MKRVIDOR4000.ttf** na directoria dos *output files*, que lista em formato de texto os *bytes* que descrevem a configuração interna que a FPGA terá de tomar para que possa ser implementado e executado o projecto.

Foi necessário reverter *bit* a *bit* cada um destes *bytes* antes de enviar para a *board* MKR Vidor. Para tal foi escolhido um *software* escrito em *java*.

Este processo gerou um ficheiro **app.h** que pode por fim ser adicionado como *header* ao *sketch* vazio de Arduino dado em [31], sendo feita por fim a compilação deste no IDE e depois desta ter sucedido, o *upload* para a placa.

A implementação na *breadboard*, vista na figura seguinte 4.19, traduziu-se como referido em usar um interruptor impor controlo ao input "ival". A resistência de 330 Ω entre o interruptor e o *ground* foi colocada para evitar a ocorrência de um curto circuito, e a de 2.2 kΩ no LED é para limitar a corrente que nele circula, sendo esta entre 5 e 10 mA.

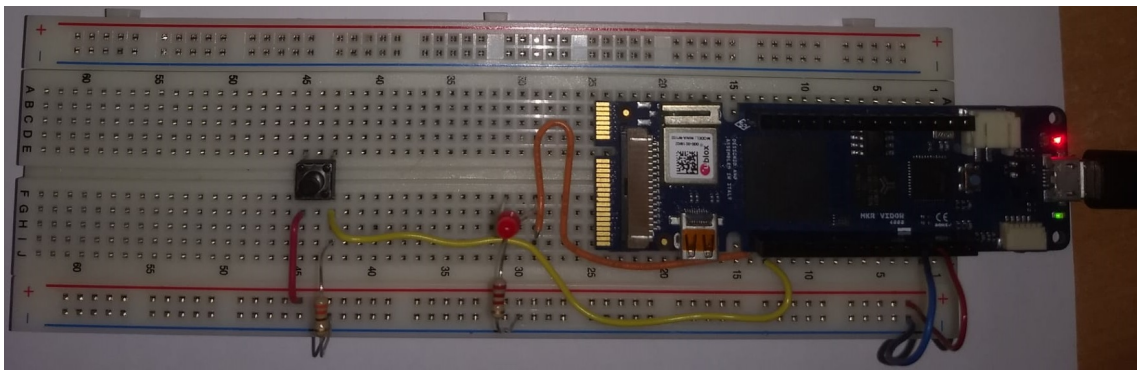

Figura 4.19: Implementação do *blink* com um interruptor na *breadboard*

4.4.1 Esquemático *VS* HDL

Acabámos de ver como, usando a ferramenta de esquemáticos do Quartus, podemos especificar módulos bastante simples, sendo que este método apresenta como vantagens:

- componente visual conferindo maior acessibilidade;
- construção de diagramas electrónicos usando simples funções lógicas como AND / NOT / OR, *gates*, *flip-flops*, etc;
- interconectividade entre módulos maiores.

Por outro lado, esta metodologia não pode ser considerada se for necessário especificar sistemas mais complexos ou mesmo pequenas máquinas de estado.

4.5 Implementação de um filtro em Quartus

4.5.1 Elementos dos Circuitos

Nesta secção é apresentada a implementação esquemática de um filtro FIR na sua forma directa. Como visto na subsecção 2.4.4, um filtro FIR digital consiste em apenas três elementos digitais de circuitos, que são o multiplicador (*multiplier*), somador (*adder*) e o atraso (*delay*).

Portanto, foram escolhidos esquemáticos de blocos da biblioteca de *software* do Quartus para implementar estes elementos.

4.5.1.1 *ALTMEMMULT* como multiplicador

Para desempenhar o papel de multiplicador, foi seleccionado o bloco *ALTMEMMULT*.

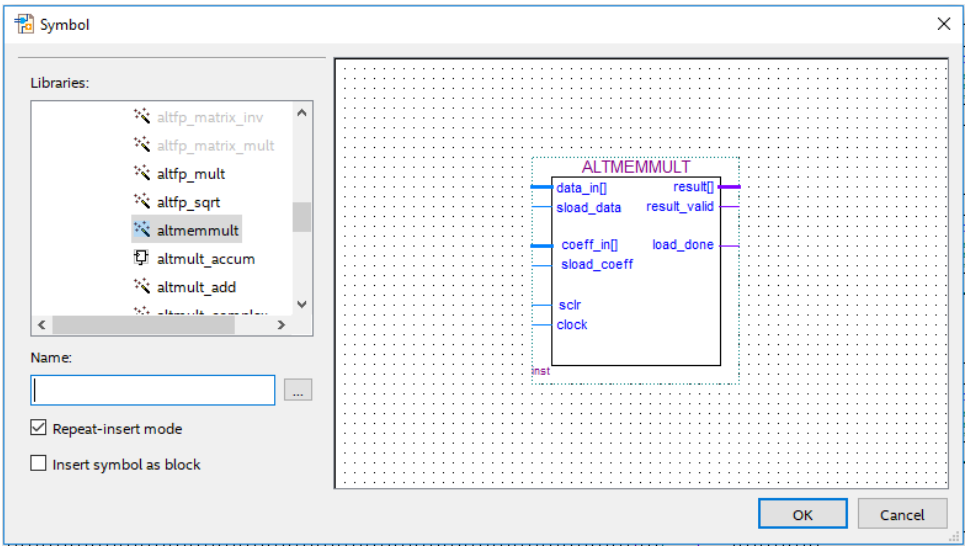

Figura 4.20: Bloco *ALTMEMMULT* disponível na biblioteca

A mega função aritmética *ALTMEMMULT* implementa um multiplicador e oferece muitos recursos adicionais, que incluem: suporte para representação de dados tanto *signed* como *unsigned*, opções para implementação em blocos de memória, suporte para *pipelining* com latência de saída parametrizada, *synchronous-clear* e *inputs* de controle de carga e por fim, validação de resultados.

Para uso deste trabalho, o *ALTMEMMULT* foi alterado em três blocos diferentes, para corresponder aos cinco (sendo que estes são espelhados) coeficientes mostrados na imagem 4.3, ficando com 151, 223, 252, 223 e 151.

Externamente são todos semelhantes, apenas desenhados para multiplicarem por um coeficiente diferente internamente, na figura 4.21 encontra-se o primeiro.

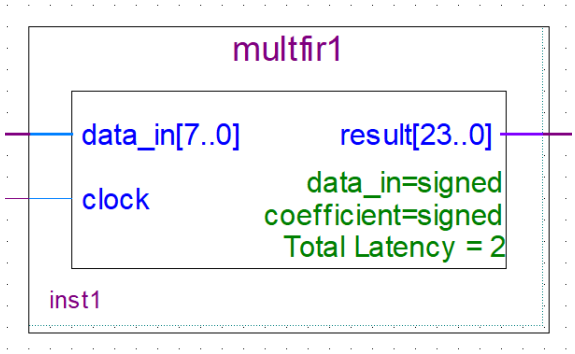

Figura 4.21: *Altemmult* final a ser usado

4.5.1.2 *PARALLEL_ADD* como somador

Para desempenhar o papel de multiplicador, foi seleccionado o bloco *PARALLEL_ADD*.

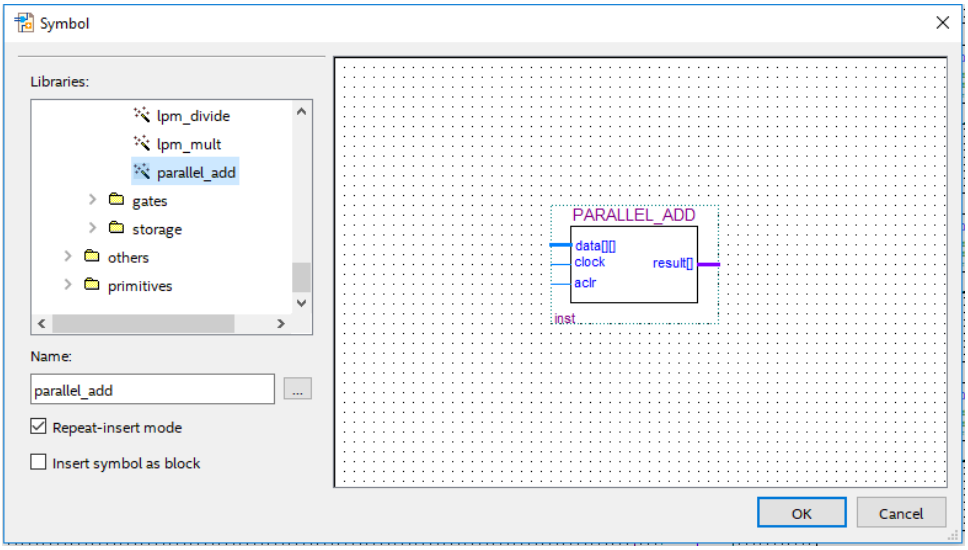

Figura 4.22: *PARALLEL_ADD*

A mega função *PARALLEL_ADD* permite seleccionar o número de entradas a serem adicionadas para produzir um resultado de soma único.

É possível adicionar ou subtrair mais de dois operandos e fazer um *shift* automático aos operandos de entrada ao entrar na função. Esta mudança de operandos de entrada é útil para estruturas de filtro FIR que exigem mudança e acumulação de produtos parciais, sendo assim um método eficiente de os implementar.

As suas aplicações incluem os filtros FIR em série, contadores e outras operações que envolvam dados em fluxo serial que devem ser adicionados durante um determinado período de tempo. Para uso neste trabalho, o *PARALLEL_ADD* foi alterado de forma a permitir a soma de cinco *buses* de dados para implementar os filtros projectados.

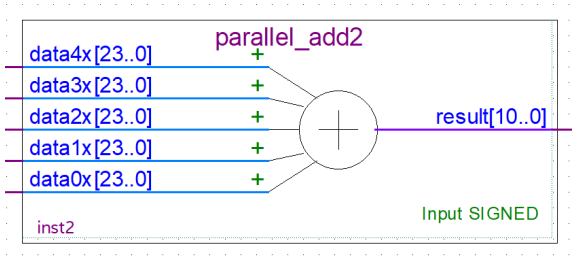

Figura 4.23: *PARALLEL_ADD* final a ser usado, com possibilidade de somar 5 *buses*

4.5.1.3 *74273b D-flip-flop* como elemento de atraso

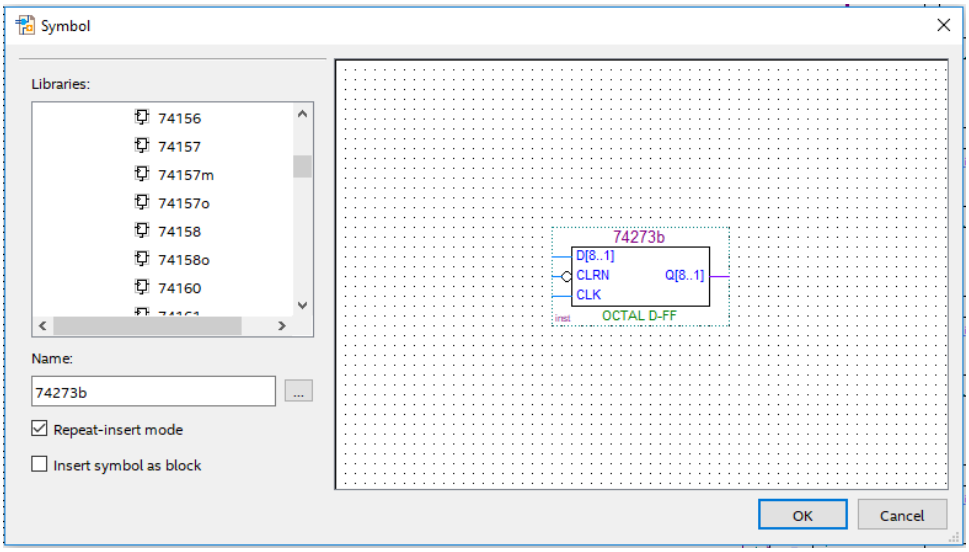

Figura 4.24: *74273b D-flip-flop*

Utilizam circuitos TTL para implementar o tipo D de lógica *flip-flop* com uma entrada clara directa. De seguida, foi implementado numa fase inicial a arquitectura de forma directa.

4.5.2 Junção dos elementos - implementações das arquitecturas

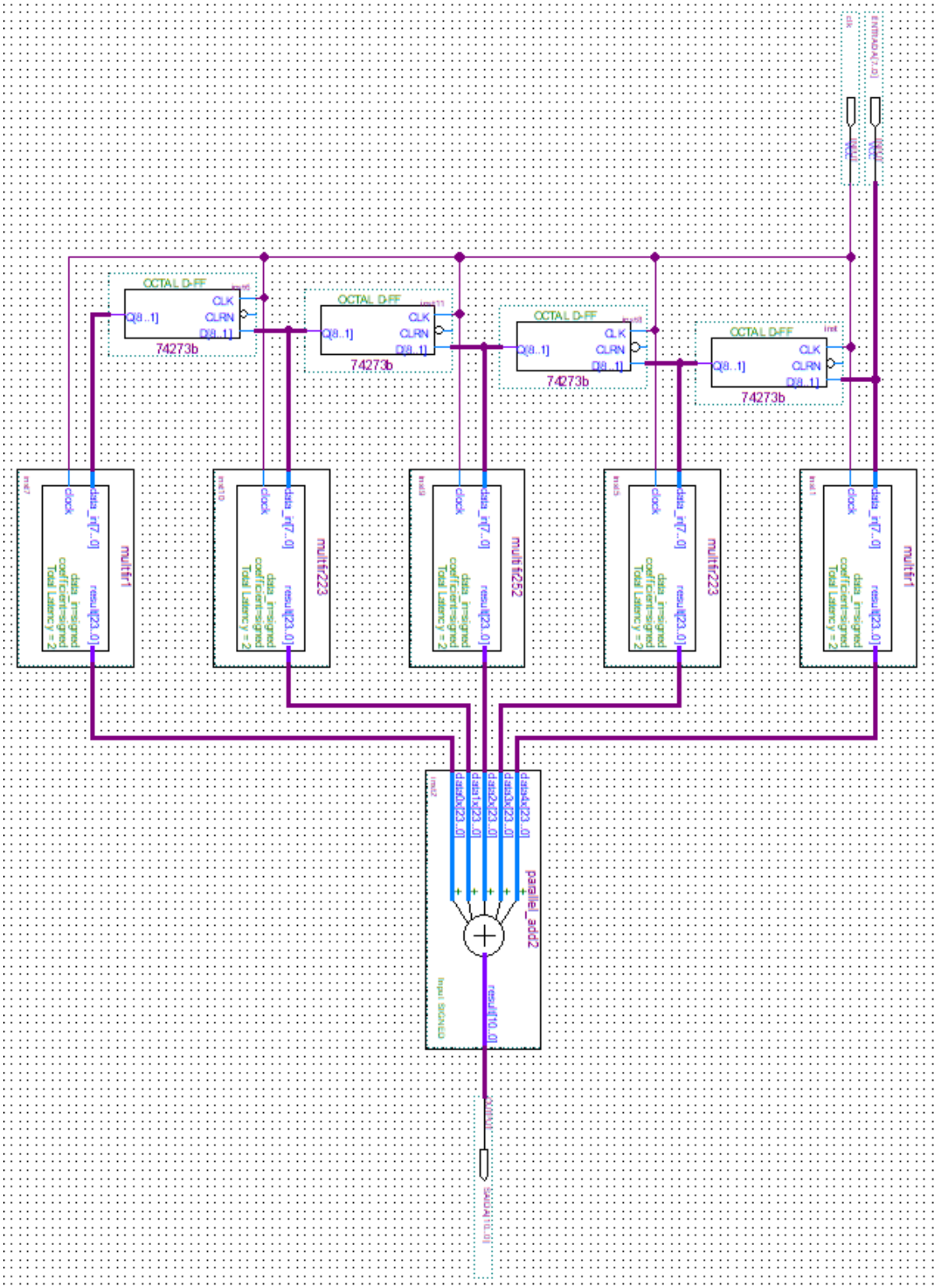

Figura 4.25: Esquemático de um filtro FIR ordem 4 de forma directa no Quartus

Usando então os blocos como elementos de circuito digital mostrados na subsecção 4.5.1, um filtro digital FIR pode ser implementado esquematicamente para qualquer comprimento desejado, neste caso N = 5 e de quarta ordem. É possível observar nas imagens 4.31, 4.26 e 4.27 as implementações tomadas para as arquitecturas propostas.

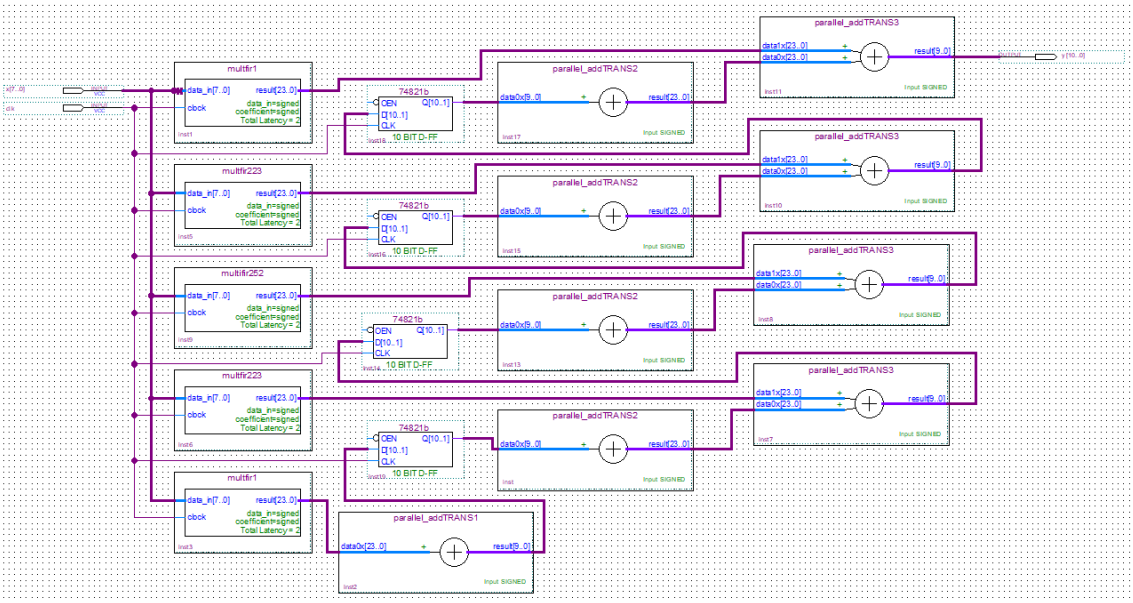

Figura 4.26: Esquemático de um filtro FIR ordem 4 de forma directa transposta no Quartus

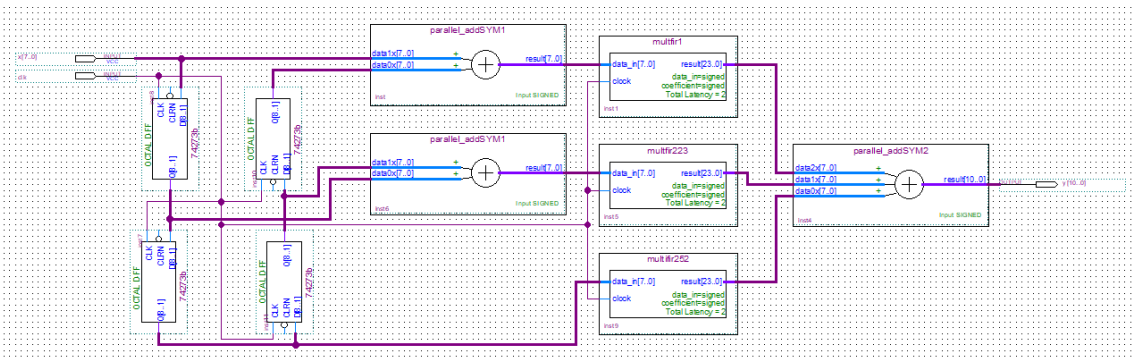

Figura 4.27: Esquemático de um filtro FIR ordem 4 de forma directa simétrica no Quartus

Nas arquitecturas de forma directa transposta e forma directa simétrica representadas nas figuras 4.26 e 4.27 respectivamente, foi necessário alterar novos blocos de *PARALLEL_ADD* de forma a poder fazer menos adições, dependendo da estrutura, e ajustar o número de bits dos *buses* de entrada devido à ordem alterada das operações.

Foi também usado uma variante do *flip-flop* de forma a acomodar um *bus* maior, o *74821b*, visto que na forma directa transposta do atraso no tempo Z^{-1} vem depois da multiplicação pelos coeficientes.

4.5.3 Compilação das arquitecturas

Tal como na secção 3.4 em que se refere o trabalho feito em [16], podemos observar que os resultados obtidos através da compilação feita no ambiente Quartus Prime comprovam as mesmas observações de baixo nível já tidas no artigo. Nas figuras 4.28, 4.29 e 4.30 observam-se os resultados das compilações das estruturas da forma directa, directa simétrica e directa transposta respectivamente. É possível ter assim dados para o número de elementos lógicos e de *bits* de memória que cada arquitectura usa para N = 5.

Com um aumento ao número do comprimento do filtro, teria sido possível confirmar todos os valores das tabelas 3.1, 3.2 e 3.3, tal não foi implementado devido à incerteza do material conseguir suportar filtros tão grandes e da incapacidade de depois serem testados na *breadboard*, como é feito na secção seguinte 4.5.4.

Flow Summary

<<Filter>>

Flow Status	Successful - Wed Sep 18 16:03:41 2019
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	Filtros_pre_placa
Top-level Entity Name	Ord5_FIR_direct
Family	Cyclone 10 LP
Total logic elements	76 / 6,272 (1 %)
Total registers	32
Total pins	20 / 89 (22 %)
Total virtual pins	0
Total memory bits	14,080 / 276,480 (5 %)
Embedded Multiplier 9-bit elements	0 / 30 (0 %)
Total PLLs	0 / 2 (0 %)
Device	10CL006YE144C6G
Timing Models	Final

Figura 4.28: Compilação do filtro FIR de forma directa

Flow Summary

<<Filter>>

Flow Status	Successful - Wed Sep 18 16:13:36 2019
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	Filtros_pre_placa
Top-level Entity Name	FIRsym
Family	Cyclone 10 LP
Total logic elements	54 / 6,272 (< 1 %)
Total registers	32
Total pins	20 / 89 (22 %)
Total virtual pins	0
Total memory bits	8,448 / 276,480 (3 %)
Embedded Multiplier 9-bit elements	0 / 30 (0 %)
Total PLLs	0 / 2 (0 %)
Device	10CL006YE144C6G
Timing Models	Final

Figura 4.29: Compilação do filtro FIR de forma directa simétrica

Flow Summary

<<Filter>>

Flow Status	Successful - Wed Sep 18 16:08:14 2019
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	Filtros_pre_placa
Top-level Entity Name	FIRtrans
Family	Cyclone 10 LP
Total logic elements	51 / 6,272 (< 1 %)
Total registers	40
Total pins	20 / 89 (22 %)
Total virtual pins	0
Total memory bits	13,056 / 276,480 (5 %)
Embedded Multiplier 9-bit elements	0 / 30 (0 %)
Total PLLs	0 / 2 (0 %)
Device	10CL006YE144C6G
Timing Models	Final

Figura 4.30: Compilação do filtro FIR de forma directa transposta com N = 5 no Quartus Prime

4.5.4 *Set up* da *breadboard*

O desafio inerente da montagem da MKR Vidor 4000 e a cablagem envolvente numa *breadboard* para testes foi o do tamanho reduzido desta, que visto ser tão limitado, não existe nela qualquer tipo de LEDs que possam servir como *outputs* visuais, comparativamente a outras placas com FPGAs.

Excluindo o pequeno LED que está embutido em todos os dispositivos Arduino e que activa no programa "blink", demonstrado na secção 4.4, não existe nada que se possa fazer sem recorrer a material externo.

Coloca-se assim a seguinte problemática de: como testar? O que usar como *inputs* e *outputs* para poder correr o projecto e ver o resultado desejado da filtragem de uma forma inequívoca?

Uma vez que a placa tem também um número reduzido de *pins* que podem ser usados para fazerem ligações externas, houve receio de que devido à sensibilidade inerente ao material, uma ligação incorrecta fosse comprometer a MKR Vidor e o trabalho feito.

Além da sensibilidade, levantou-se também a questão de também serem precisos *inputs* externos, neste caso tendo sido decidido usar o mesmo *switche* de 4.4, para simular os parâmetros de entrada a serem filtrados. Levantou-se a preocupação de haver espaço em tanto *hardware* como *software* para todas as ligações externas novas numa placa tão reduzida.

Listagem 4.4: Selecção dos *pins* a usar

```
1  // SAM D21 PINS
2  inout         bMKR_AREF,
3  inout  [6:0]  bMKR_A,
4  inout  [14:0] bMKR_D,
```

Depois de uma leitura atenta no *source code* do ficheiro de topo, foi possível reparar num número (reduzido) mas suficiente de *pins* prontos a ser utilizados, mais do que esperado: 7 *pins* bMKR_A e 15 bMKR_D. Como pode ser visto na listagem 4.4, eles são do tipo *inout*, logo podem ser usados de ambas as formas.

Confirmando depois nos esquemáticos da literatura da MKR Vidor, foi visto que todos eles eram "seguros" de se usar, sendo que o maior perigo à integridade do material é mudar no ficheiro Verilog a tensão e a direcção dos I/O *pins*.

A implementação final, demonstrada na imagem seguinte 4.31, foi tida com o maior cuidado e usando as mesmas resistências que na secção 4.4, no exemplo da montagem "blink". O LED verde juntamente com o fio da mesma cor serviu apenas como ponta de prova e não tem relevância à montagem. A montagem dos LEDs vermelhos aloca assim cada um dos *bits* do *bus* de saída dos esquemáticos, declarados dentro do ficheiro de topo Verilog, do mais ao menos significativo vendo da esquerda para a direita.

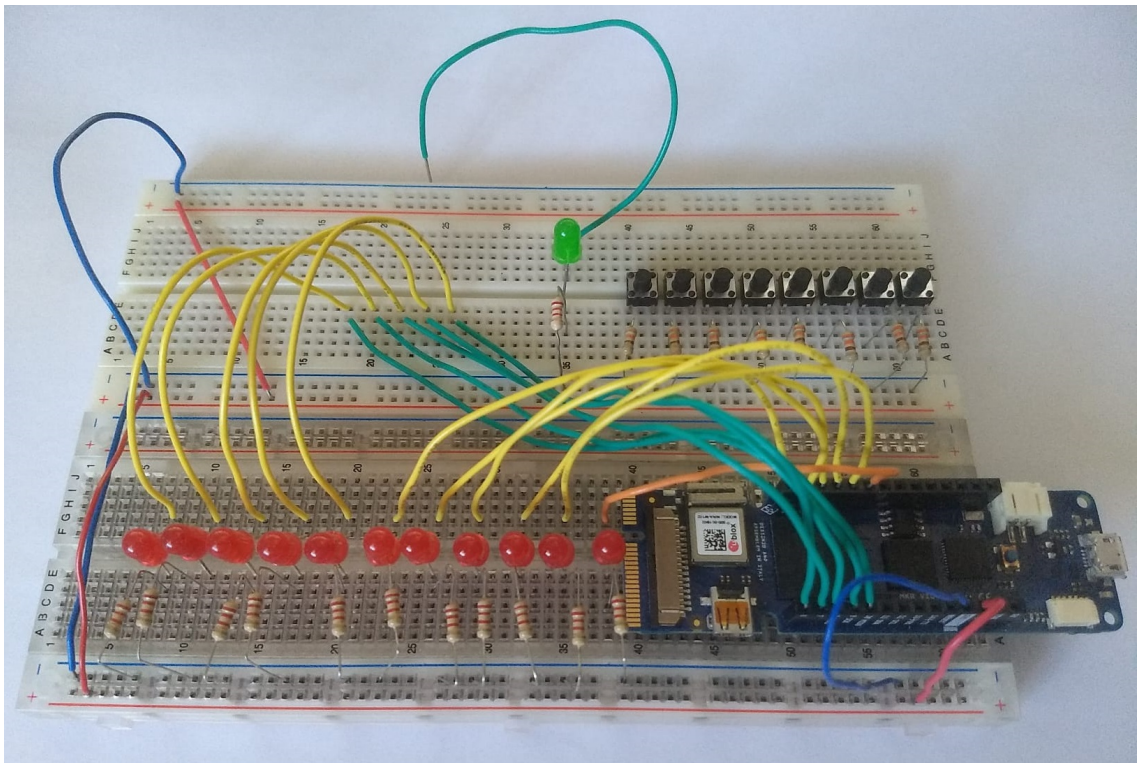

Figura 4.31: Montagem da MKR Vidor 4000 numa *board* com ligações para o *display* dos *outputs* em forma de LED

Caso não houvesse em número suficiente (sendo necessários para testar 11 *bits* de saída para 8 de entrada), uma alternativa seria projectar o *output* num ecrã, talvez através de uma ligação HDMI. Devido a complicações com a integração dos sinais tanto de entrada como de saída da FPGA através Arduino IDE, tal não foi possível concretizar no período alocado de tempo, mas também não foi necessário.

4.6 Simulações

Na secção anterior, foi demonstrado o funcionamento da tecnologia e as limitações encontradas durante o desenvolvimento que limitaram um teste mais aprofundado ao trabalho.

Apesar de ter sido possível verificar a robustez dos filtros, tornou-se impossível devido à inacessibilidade do material fazer uma comparação directa com outras soluções encontradas na literatura, sendo que a implementação e resultados apresentados padecem de um rigor que seria colmatado apenas com ou uma repetição do trabalho num intervalo maior de tempo, de forma a permitir um maior suporte às ferramentas de *hardware* usado, ou também com o uso de uma FPGA diferente.

Deste modo, este capítulo foca-se numa análise qualitativa dos resultados obtidos, tanto em ambiente de simulação como real, bem como avaliar a qualidade e limitações do trabalho desenvolvido.

4.6.1 Simulações do Passa Baixo em Ambiente Quartus

O método de simulação usado nesta secção é baseado no desenho de formas de onda, semelhantes aos diagramas de tempo, que permitem uma representação de um conjunto de sinais eléctricos no domínio do tempo, sendo dados como entradas para uma ferramenta de simulação.

As saídas do simulador são, também em formato de formas de onda.

A ferramenta usada foi a Simulation Waveform Editor (SWE), que está disponível para uso com o *software* Quartus II da Altera versão 13.0 ou posterior.

Permite ao utilizador aplicar entradas ao circuito projectado, geralmente chamado de vectores de teste, na forma de formas de onda e observar as saídas geradas em resposta. Como se pode observar na figura seguinte 4.32, o filtro desenhado comporta-se da mesma forma que o filtro projectado em Matlab, na secção 4.1.1.

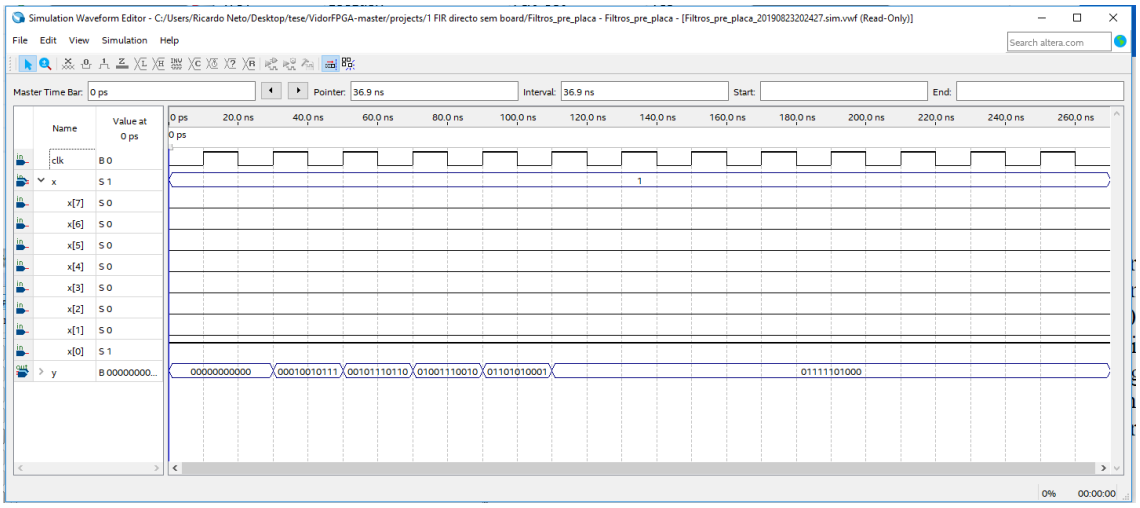

Figura 4.32: Resultado da simulação para a estrutura FIR de forma directa para o filtro projectado com N = 5 no Quartus Prime - em binário

Depois de seleccionados os nós de entrada e saída a serem carregados para o SWE com a opção *Node Finder*, é então inserido no bus de entrada, x[], o valor 00000001 (colocando apenas x[0] com o valor 1 e o resto dos bits a 0, como é possível ver), ou seja 1 em decimal, assim como foi feito anteriormente na secção 4.1.1.

O *clock* usado foi um de valor semelhante ao do *clock* interno da FPGA, 80MHz, de forma a simplificar esta implementação.

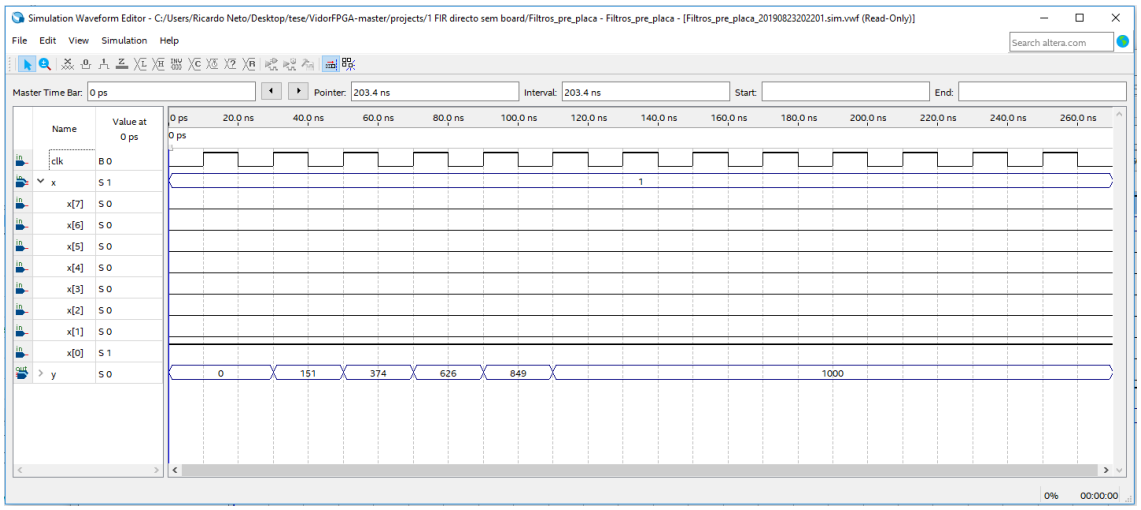

Figura 4.33: Resultado da simulação para a estrutura FIR de forma directa para o filtro projectado com N = 5 no Quartus Prime - em decimal

É possível observar que o valor de *steady state* em y[], o bus de saída, é 1111101000, que é efectivamente 1000. Como confirmação final, os valores de saída y[] foram convertidos em decimal para não deixar dúvidas na simulação, como se pode conferir na figura anterior 4.33.

Todos os valores correspondem aos do sinal filtrado para exemplo de simulação do filtro projectado para f_S=1000 Hz, da secção 4.1.1

4.7 Simulações do Passa Baixo na *board* com a MKR Vidor

Nesta fase final de testes com a placa na *breadboard*, foi tomado um ligeiro atalho. Visto que, como referido na subsecção 4.5.4, através do Verilog e pela Arduino IDE não houve sucesso em interagir com os *buses* de entrada de uma forma externa, foram forçados os valores na instância declarada do filtro, de maneira a garantir que o valor de *input* era o desejado.

Listagem 4.5: Instância declarada a forçar os bits de entrada

```
Ord5_FIR_direct Ord5_FIR_direct_inst
(
  .clk (wOSC_CLK),

  //.ENTRADA[0](bMKR_A[0]),
  //.ENTRADA[1](bMKR_A [1]),
  //.ENTRADA[2](bMKR_A [2]),
  //.ENTRADA[3](bMKR_A [3]),
  //.ENTRADA[4](bMKR_A [4]),
  //.ENTRADA[5](bMKR_A [5]),
  //.ENTRADA[6](bMKR_A [6]),
  //.ENTRADA[7](bMKR_D [11]),

  //.SAIDA[0] (bMKR_D [0]),
  //.SAIDA[1] (bMKR_D [1]),
  //.SAIDA[2] (bMKR_D [2]),
  //.SAIDA[3] (bMKR_D [3]),
  //.SAIDA[4] (bMKR_D [4]),
  //.SAIDA[5] (bMKR_D [5]),
  //.SAIDA[6] (bMKR_D [6]),
  //.SAIDA[7] (bMKR_D [7]),
  //.SAIDA[8] (bMKR_D [8]),
  //.SAIDA[9] (bMKR_D [9]),
  //.SAIDA[10] (bMKR_D [10]),

  //7'b1 = 00000001
  .ENTRADA(7'b1),

  //.ENTRADA({bMKR_D[11],bMKR_A[6:0]}),
  .SAIDA(bMKR_D[10:0])
);
```

Na variável "ENTRADA", o x[] anterior, foi declarado valor de 7'b1, ou seja o mesmo 00000001 dos testes anteriores vistos nas figuras 4.32 e 4.33 . Para tal foram retirados os *switches* da montagem, ou seja o input foi fixo e estes obsoletos.

Os bits que se encontram acessos formam um 01111101000, como visto na figura seguinte 4.34 - ou seja 1000 em decimal, tal como foi obtido nas simulações obtidas na secção anterior 4.7.

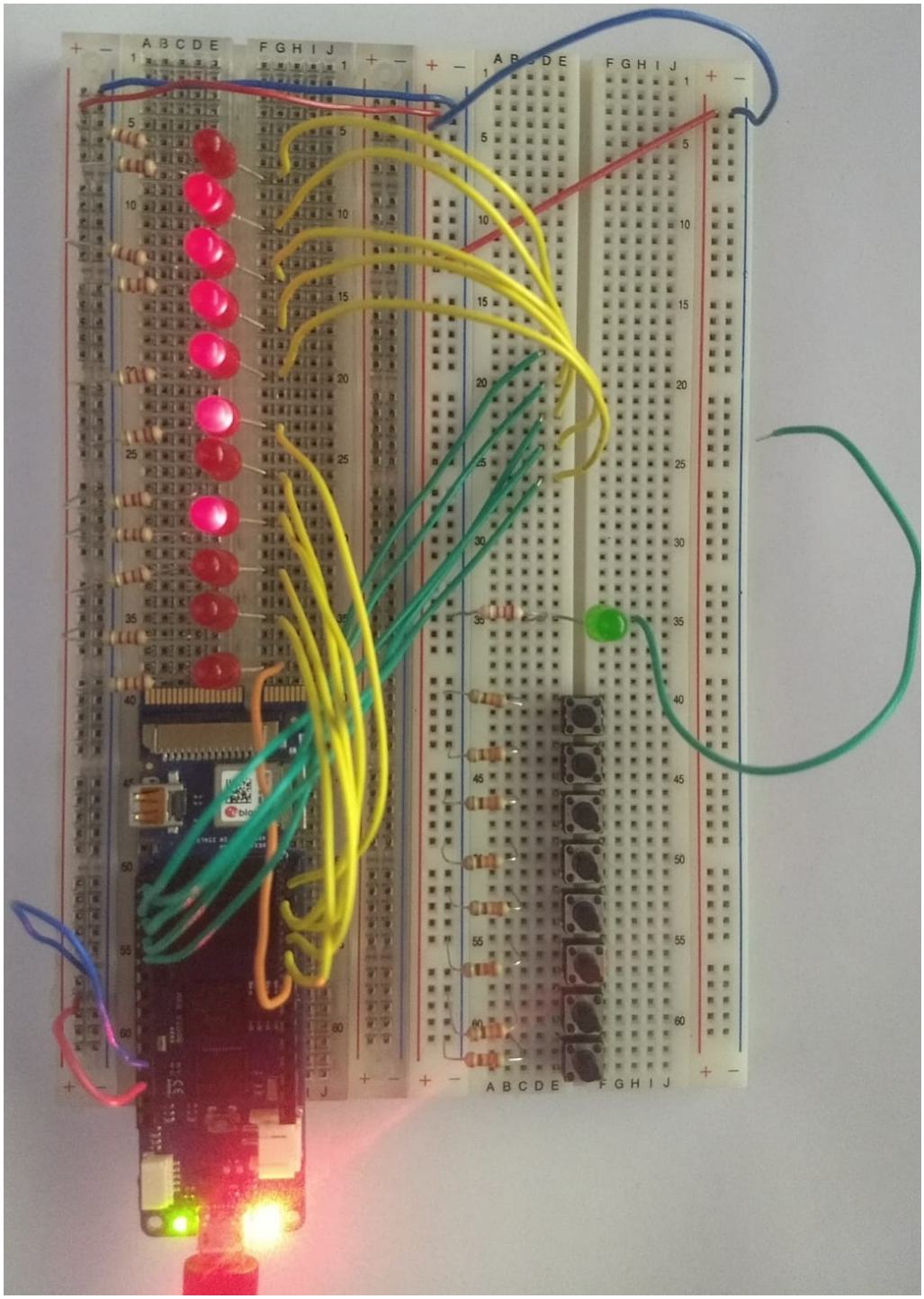

Figura 4.34: Resultado obtido correspondente ao esperado, *uploaded* na placa

Capítulo 5

Conclusões e Trabalho Futuro

Neste estudo, implementações esquemáticas em FPGA para três estruturas fundamentais diferentes de filtros digitais FIR são investigadas, para posteriormente serem adaptadas num contexto de processamento de imagem.

De forma a obter um resultado de referência para as comparações, recorreu-se a simulações no Matlab para o comprimento do filtro N = 5 para um sinal de entrada designado. Por fim, as três estruturas FIR são implementadas esquematicamente no Quartus Prime para N = 5 e depois na placa inserida na *breadboard*. Visto que os resultados dos testes executados em ambos os ambientes são os mesmos do Matlab, tal significa que as implementações esquemáticas propostas foram bem-sucedidas.

O trabalho desta dissertação foi marcado, como referido anteriormente, por limitações em termos de apoio relativamente ao material escolhido, que aquando da escrita desta conclusão continua estático em termos de suporte e sem quaisquer *updates* para breve. Apesar da "falsa" acessibilidade apresentada placa Arduino MKR Vidor 4000, a implementação de um filtro FIR que pode ser alterado para qualquer especificações foi um sucesso e pode efectivamente continuar a ser desenvolvido.

Não foi prosseguido com filtros de ordens maiores pois o foco do trabalho seria numa implementação de um algoritmo de processamento de imagem - algo que acabou por não suceder. Isto devido à quantidade de trabalho que tal revelou ser para o tempo e material disponível, que juntando à já inacessibilidade da tecnologia se traduziria num trabalho extra de duas dissertações.

Esta dissertação apresenta-se assim como um estudo compreensivo sobre filtros e a sua respectiva implementação numa Arduino MKR Vidor 4000, com o processamento a ter ficado aquém mas não sendo impossível - apenas não com esta placa e não neste intervalo de tempo.

5.1 Trabalho Futuro

De acordo com o tempo limitado definido para a realização desta dissertação, neste trabalho os filtros foram implementados e testados apenas em um só fabricante. Seria interessante abranger mais fabricantes, a fim de determinar qual abordagem e ambiente que implementaria o filtro FIR de forma mais eficaz em relação aos cenários apresentados neste trabalho.

No futuro com um maior domínio sobre as HDL e mais escrutínio relativamente ao material escolhido, seria possível implementar de facto um algoritmo de processamento de imagem numa FPGA recorrendo a filtros. Possivelmente numa FPGA da gama Xilinx, que foram estudadas em literatura paralelamente a este trabalho.

ERROR: File 'chapter6' does not exist!!!

Bibliografia

[1] P. ICNF/MA-MAFDR. *Incêndios rurais e área ardida – Continente*. Acedido em: 2019-09-10. URL: https://www.pordata.pt/Portugal/Inc%c3%aandios+rurais+e+%c3%a1rea+ardida+%e2%80%93+Continente-1192.

[2] P. SGIF. *"...a área ardida em Portugal este ano é inferior em 26 por cento à de 2018. Confirma-se?"* Acedido em: 2019-01-10. URL: https://poligrafo.sapo.pt/fact-check/eduardo-cabrita-diz-que-a-area-ardida-em-portugal-este-ano-e-inferior-em-26-a-de-2018-confirma-se.

[3] D. Jones. *EEVblog Number 496 - What Is An FPGA?* Acedido em: 2019-02-14. URL: https://www.eevblog.com/forum/blog/eevblog-496-what-is-an-fpga/.

[4] A. O. Agerholm. *In the Age of FPGA*. Acedido em: 2019-02-20. URL: https://www.napatech.com/age-of-fpga/.

[5] D. Bradley. *Keeping up with Moore's Law*. Acedido em: 2019-02-16. URL: https://phys.org/news/2018-12-law.html.

[6] A. Akif. "Elektroteknik FIR Filter Features on FPGA FIR Filter Finesser på FPGA". Em: 2018.

[7] A. Ehliar e D. Liu. "An ASIC perspective on FPGA optimizations". Em: *2009 International Conference on Field Programmable Logic and Applications*. 2009, pp. 218–223. DOI: 10.1109/FPL.2009.5272311.

[8] J. C. R. Amos R. Omondi. *FPGA Implementations of Neural Networks*. Springer, Boston, MA, 2006. DOI: 10.1007/0-387-28487-7. URL: https://doi.org/10.1007/0-387-28487-7.

[9] R. Dekker. *What's the Difference Between VHDL, Verilog, and SystemVerilog?* Acedido em: 2019-08-10. URL: https://www.electronicdesign.com/what-s-difference-between/what-s-difference-between-vhdl-verilog-and-systemverilog.

[10] Philippe. *Arduino MKR Vidor 4000 – Présentation et mise en route*. Acedido em: 2019-04-17. URL: https://systemes-embarques.fr/wp/archives/arduino-mkr-vidor-4000-presentation-et-mise-en-route/.

[11] S. Liebson. *Quickly and Easily Apply FPGAs with the Arduino MKR Vidor 4000*. Acedido em: 2019-02-09. URL: https://www.digikey.pt/en/articles/techzone/2018/nov/quickly-easily-apply-fpgas-arduino-mkr-vidor-4000.

[12] Arduino, LLC. *ARDUINO MKR VIDOR 4000*. Acedido em: 2019-02-09. URL: https://store.arduino.cc/mkr-vidor-4000.

[13] A. Antoniou. *Digital Signal Processing*. 2nd. Mcgraw-Hill, 2016. ISBN: 0071846034, 9780071846035.

[14] Jyoti, A. Kumar e A. Sangwan. "Designing of FIR Filter Using FPGA: A Review". Em: *Nanoelectronics, Circuits and Communication Systems*. Ed. por V. Nath e J. K. Mandal. Singapore: Springer Singapore, 2019, pp. 493–505. ISBN: 978-981-13-0776-8.

[15] C. Hu. "Design and verification of FIR filter based on Matlab and DSP". Em: *2012 International Conference on Image Analysis and Signal Processing*. 2012, pp. 1–4. DOI: 10.1109/IASP.2012.6425042.

[16] O. Coşkun e K. Avci. "FPGA Schematic Implementations and Comparison of FIR Digital Filter Structures". Em: *Balkan Journal of Electrical and Computer Engineering* (fev. de 2018). DOI: 10.17694/bajece.369234.

[17] C. R. Chou, S. Mohanakrishnan e J. B. Evans. "FPGA IMPLEMENTATION OF DIGITAL FILTERS". Em: 1993.

[18] E. Kolawole, W. Ali, P Cofie, J. Fuller, C Tolliver e P. Obiomon. "Design and Implementation of Low-Pass, High-Pass and Band-Pass Finite Impulse Response (FIR) Filters Using FPGA". Em: *Circuits and Systems* 06 (jan. de 2015), pp. 30–48. DOI: 10.4236/cs.2015.62004.

[19] V. Varshney e M. Tiwari. "Realization of an FIR filter using ATMEGA32 microcontroller". Em: *2017 International Conference on Emerging Trends in Computing and Communication Technologies (ICETCCT)*. 2017, pp. 1–4. DOI: 10.1109/ICETCCT.2017.8280325.

[20] I. Thingom e P. Khundrakpam. "FPGA implementation of FIR filter using RADIX-2r". Em: *2016 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*. 2016, pp. 1524–1528. DOI: 10.1109/WiSPNET.2016.7566391.

[21] D. A. Mishra. "FIR Filter Design on Chip Using VHDL". Em: *IPASJ International Journal of Computer Science (IIJCS) Volume 2, Issue 7, ISSN 2321-5992* 2 (jul. de 2014).

[22] R Madhukar Narsale, D Gawali e A Kulkarni. "FPGA based design & implementation of low power FIR filter for ECG". Em: *Signal Process./Int. J. Sci. Eng. Technol. Res.* 3 (jan. de 2014).

[23] K. Aboutabikh e N. Aboukerdah. "Design and implementation of a multiband digital filter using FPGA to extract the ECG signal in the presence of different interference signals". Em: *Computers in Biology and Medicine* 62 (abr. de 2015). DOI: 10.1016/j.compbiomed.2015.03.034.

[24] A. Ehliar. "Performance driven FPGA design with an ASIC perspective". Em: 2009.

[25] P. Födisch, A. Bryksa, B. Lange, W. Enghardt e P. Kaever. "Implementing High-Order FIR Filters in FPGAs". Em: *CoRR* abs/1610.03360 (2016). arXiv: 1610.03360. URL: http://arxiv.org/abs/1610.03360.

[26] P. K. Meher, S. Chandrasekaran e A. Amira. "FPGA Realization of FIR Filters by Efficient and Flexible Systolization Using Distributed Arithmetic". Em: *IEEE Transactions on Signal Processing* 56.7 (2008), pp. 3009–3017. ISSN: 1053-587X. DOI: 10.1109/TSP.2007.914926.

[27] MATLAB©. *Products and Services*. Acedido em: 2019-04-17. URL: https://www.mathworks.com/products.html?s_tid=gn_ps.

[28] J. Lewis. *Massimo introducing MKR 4000*. Acedido em: 2019-08-16. URL: https://www.baldengineer.com/mkr-vidor-4000-brings-fpga-to-makers.html/mkr-vidor-4000-visual-fpga-editor#main.

[29] Arduino. *Getting Started with the Arduino MKR Vidor 4000*. Acedido em: 2019-01-10. URL: https://www.arduino.cc/en/Guide/MKRVidor4000.

[30] Intel. *Download Center for FPGAs*. Acedido em: 2019-04-17. URL: https://www.intel.com/content/www/us/en/programmable/downloads/download-center.html.

[31] D. Pennisi. *repository for Vidor FPGA IP blocks and projects*. Acedido em: 2019-01-10. URL: https://github.com/vidor-libraries/VidorFPGA.